Autonomous Navigation and Perception – Final Project

Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty

Submitted by:

Eli Goldenshluger – 310237466

Nitzan Makmal - 203359427

Introduction and overview

This paper is about a sampling based algorithm - Rapidly Random Belief Trees (RRBT) which was introduced in the paper "Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty" by Adam Bry and Nichols Roy in ICRA (IEEE International Conference on Robotics and Automation)*2011* . Adam Bry is an MIT graduate that worked for google and amazon and now is a CEO of skydio – a company that is pioneering in autonomous aerial systems , AI and computer vision.

The main problem presented in the article is a scenario of an autonomous system or a robot which navigates in an unknown environment. The system must reach some designated goal area while assuring obstacle avoidance and doing all of this in the shortest path possible while generating an optimal trajectory. Naturally, all of the above must be done online, while using only on board sensors, meaning, we have no any prior information about the environment.

In general, our problem consists of two main issues. the first issue is online path planning, this issue can be tackled by any sampled base path planning algorithm.

The main idea of sampling based algorithms is randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the state space. The second issue is dealing with uncertain dynamics and observations, which the samples based algorithms are not designed to deal with.

In the scope of our course – Autonomous Navigation and Perception, we learned about number of solutions to a similar set of problems. When trying to face the problem we can implement RRT or RRT* - Rapidly exploring random trees. In these algorithms we generate a random node in each iteration of the algorithm, by this exploring the environment and constructing a graph while avoiding obstacles until the goal is reached. Those algorithms converge better as number of samples increased, or in other words they converge to the optimal path when approaching infinite number of samples/iterations. The problem with these algorithms starts when we introduce stochasticity into the motion and the observation model. As those algorithms are intended to work in a fully deterministic scenario, the stochasticity is not taken into account, and it may result in an obstacle collision or divergence from the desired path/goal area. In the example given in fig (1):

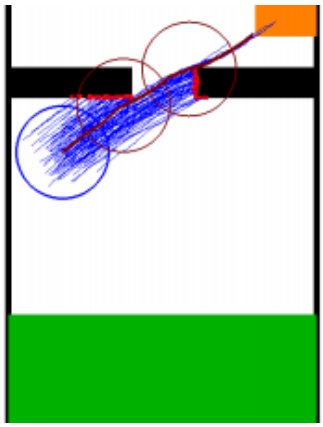

Figure 1

we can see that when executing actuations resulted from a deterministic settings an obstacle collision is likely to take place. The red path is generated by a sampling algorithm operating on a deterministic model. The blue ellipses represents the uncertainty of the given state. The blue

lines represents all the possible trajectories throughout this path. we can see that a collision may occur and full obstacle avoidance is not achieved in some cases.

In our course we also talked about path planning in the belief space – one algorithm that was discussed is the BRM - which projects the sampling based search into the belief space, choosing the path at the minimum uncertainty at the goal. Belief road mapping Takes into account stochastic motion model and observation. The BRM Assumes that the mean of the system is fully controllable – this assumption is only valid for a vehicle driving slowly and conservatively. Another assumption is that the mean of the system is fully controllable at each time step - the controller is always capable of driving the state estimate back to planned desired path. As we can see on the fig (2)

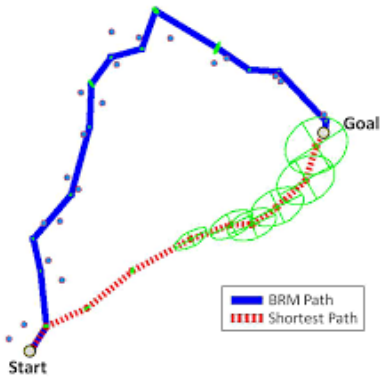

Figure 2

the red path is an output of a deterministic algorithm that chooses the shortest path but the uncertainty is greater than the goal uncertainty resulted from the BRM as demonstrated by the blue line.

The Proposed Algorithm : RRBT Algorithm – Rapidly exploring random belief trees

The paper presents the RRBT - a Sampling based motion planning algorithm, which uses Incremental sampling refinement in the present of Stochasticity while Dynamic constraints are considered. In each iteration it finds an optimal path in a small ball and therefore refines the optimal trajectory. When dealing with dynamic constraints we are not assuming perfect controls.

Preliminary material and problem formulation:

The problem formulation that the RRBT deals with:
We have a nontrivial and uncertain dynamics, meaning that our ability to predict the evolution of state over time is limited – hard dynamic constraints. We have measurement uncertainty. Another important point is the availability of the information, because in our case we have only on-board sensors (for example a self driving car using a LIDAR sensor) those sensors are state - depended. For example we can see in figure (3)

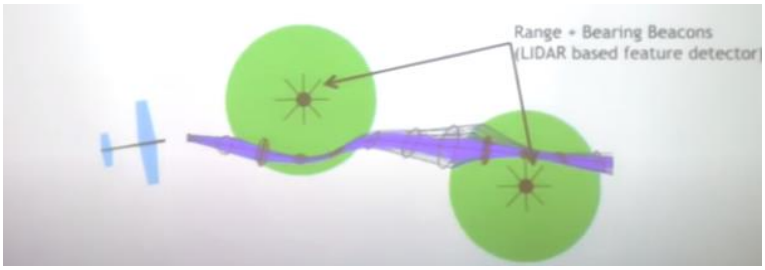

Figure 3

A LIDAR sensor that is attached to an autonomous fixed wing vehicle, the green regions symbolize information reach environments. As we leave the green areas the uncertainty of the estimation rises.

Problem formulation:

The general motion model and observation model are presented with gaussian distribution with random noise as follows:

$$(1)\quad x_t = f(x_{t-1}, u_{t-1}\,,\, \omega_{t-1}) \quad , \qquad \omega_t \sim N(0, Q)$$

$$(2)\quad z_t = h(x_t, v_t\,) \qquad\qquad , \qquad v_t \sim N(0, R)$$

The planning problem is specified with some uncertain knowledge of the robot's initial state, $x_0 \sim N(\hat{x}_0, \Sigma_0)$, a goal region in the environment $X_{goal} \subset X_{free}$ to which the robot wishes to travel.

The optimal path planning problem goal is to minimize an expectation a cost function, with respect to the cost accumulated by the current state. The expectation is with respect to the process and sensor noise ω_t and v_t, and minimization is over concatenated paths.

$$\text{argmin}\, \mathrm{E}\left[\sum_{t=1}^{T} C(x_t)\right]$$

The minimization is subjected to high probability of reaching a goal state which is obstacle free and low probability of collision with an obstacle. A key part in the algorithm is the Chance-constraint condition:

$$\check{x} = \hat{x}_0, P\big(\, x_T \notin X_{goal}\,\big) < \delta$$
$$\mathrm{P}(\,\mathrm{x_t} \notin \mathrm{X_{obs}}\,) < \delta. \forall t \in [0, \mathrm{T}]$$

The idea is to make sure that with consideration to the uncertainty, the state of the robot is always within a region that has higher probability to be in the X^{free} space than in the X^{obs} space. The size of the uncertainty is under our control as a design parameter.

The approach to path planning is to assume the availability of a CONNECT() function for finding a nominal trajectory and stabilizing controller between two states. As the notation $\check{X}, \check{U}, \check{K}$ represents the NOMINAL values of the state X, control input U and Control gain K. Suppose we two states x^a, x^b so we can write the following:

$$\left(\check{X}^{a,b}, \check{U}^{a,b}, \check{K}^{a.b}\right) = \left(Connect\left(x^a, x^b\right)\right)$$

$$\check{X}^{a,b} = \left(\check{x}_0, \check{x}_1, \check{x}_2, \ldots, \check{x}_{T_{a,b}}\right)$$

$$\check{U}^{a,b} = \left(\check{u}_0, \check{u}_1, \check{u}_2, \ldots, \check{u}_{T_{a,b}}\right)$$

$$\check{x}_0 = x^a, x^b = f\left(\check{x}_{T_{a,b}}, \check{u}_{T_{a,b}}, 0\right)$$

$$\check{x}_t = f(\check{x}_{t-1}, \check{u}_{t-1}, 0), \forall t \in \left[1, T_{a,b}\right]$$

$$\left(\check{X}, \check{U}, \check{K}\right) = \left(Connect(x^0, x^1)\right), \left(Connect(x^1, x^2)\right), \ldots, \left(Connect\left(x^{l-1}, x^l\right)\right)$$

The availability of such Connect function is beyond the scope of the paper. By this we assume that the control design decoupled from the path planning optimization.

In contrast of the BRM algorithm, here we can't assume the mean of the system is fully controllable – meaning our estimated state may not lay exactly on the nominal constructed path. During execution, x_t-the actual state, will not be available to compute the control input. Instead, we must use an estimate of x_t which we denote as $\widehat{x_t}$. The estimated error of the actual state from the nominal path is denoted by $\tilde{x}_t$, so we may write the following connections:

$$x_t = \check{x}_t + \tilde{x}_t$$

$$u_t = \check{u}_t + \tilde{u}_t$$

$$z_t = \check{z}_t + \tilde{z}_t$$

We calculate the partial derivatives of equations (1) and (2) to get a local linearized system for the motion model and the observation model.

(3) $\tilde{x}_t = A_t\tilde{x}_{t-1} + B_t\tilde{u}_{t-1} + \omega_t, \quad \omega_t \sim N(0, Q_t)$

(4) $\tilde{z}_t = C_t\tilde{z}_t + v_t, \qquad v_t \sim N(0, R_t)$

With these equations we propagate and update the expectations and distributions both on the estimated state and the estimation error from the nominal path using Extended Kalman filtering.

To evaluate a cost function and check the chance-constraint, we need a distribution over states that may be realized if we execute a given nominal trajectory. The Distribution of not getting an observation is denoted as $\boldsymbol{\Lambda}$, and the Distribution of on-line estimation error from nominal trajectory is denoted as $\boldsymbol{\Sigma}$.

All the following equations are brought here below to show the process of the Kalman filtering and the belief propagation of the estimation and the error between nominal and actual position.

(5) $\bar{\hat{x}}_t = A_t\hat{\hat{x}}_{t-1} + B_t\hat{\hat{u}}_{t-1}$

(6) $\bar{\Sigma}_t = A_t\Sigma_{t-1}A_t^T + Q_t$

(7) $S_t = C_t\bar{\Sigma}_t{C_t}^T + R_t$

(8) $L_t = \bar{\Sigma}_t{C_t}^T S_t^{-1}$

(9) $\hat{\hat{x}}_t = \bar{\hat{x}}_t + L_t(\tilde{z}_t - C_t\bar{\hat{x}}_t)$

(10) $\Sigma_t = \bar{\Sigma}_t - L_tC_t\bar{\Sigma}_t$

Equations 5-10 are the updates of the estimated state

(11) $$\bar{\mu}_t = E[\bar{\tilde{x}}_t] = E\left[A_t\hat{\tilde{x}}_{t-1} + B_t\hat{\tilde{u}}_{t-1}\right] = E\left[A_t\hat{\tilde{x}}_{t-1} + B_tK_t\hat{\tilde{x}}_{t-1}\right] = (A_t - B_tK_t)\mu_{t-1}$$

(12) $$\mu_t = E(\hat{\tilde{x}}_t) = E\left(\hat{\bar{\tilde{x}}}_t + L_t\left(\tilde{z}_t - C_t\hat{\bar{\tilde{x}}}_t\right)\right) = \bar{\mu}_t\mu_t = (A_t - B_tK_t)\mu_{t-1}$$

μ0 = 0 then μt = 0 ∀t

(13) $$\bar{\Lambda}_t = E\left[\left(A_K\hat{\tilde{x}}_{t-1}\right)\left(A_K\hat{\tilde{x}}_{t-1}\right)^T\right] = A_K\Lambda_{t-1}A_K^T$$

(14) $$\bar{\Lambda}_t = E\left[\hat{\tilde{x}}_t\hat{\tilde{x}}_t^T\right] = E\left[\left(\hat{\tilde{x}}_t + L_t\left(\tilde{z}_t - C_t\hat{\bar{\tilde{x}}}_t\right)\right)\left(\hat{\tilde{x}}_t + L_t\left(\tilde{z}_t - C_t\hat{\bar{\tilde{x}}}_t\right)\right)^T\right]$$
$$= \bar{\Lambda}_t + L_tS_tL_t^T = \bar{\Lambda}_t + L_tC_t\bar{\Sigma}_t = (A_t - B_tK_t)\Lambda_{t-1}(A_t - B_tK_t)^T + L_tC_t\bar{\Sigma}_t$$

Equations 11-14 represents the recursive update of the estimated error.

(15) $$\boldsymbol{P}(\boldsymbol{x_t}, \hat{\boldsymbol{x}}_t) = \boldsymbol{N}\left(\begin{bmatrix}\bar{\boldsymbol{x}}_t\\ \hat{\boldsymbol{x}}_t\end{bmatrix}, \begin{bmatrix}\boldsymbol{\Lambda_t} + \boldsymbol{\Sigma_t} & \boldsymbol{\Lambda_t}\\ \boldsymbol{\Lambda_t} & \boldsymbol{\Lambda_t}\end{bmatrix}\right)$$

For planning purposes this is what we care about (From equation 15):

$$\boldsymbol{P}(\boldsymbol{x_t}) = \boldsymbol{N}(\hat{\boldsymbol{x}}_t, \boldsymbol{\Lambda} + \boldsymbol{\Sigma})$$

It describes the distribution over trajectories as the sum of the on-line state estimation error, Σ_t, and the uncertainty that arises from not having yet taken observations, Λ_t . We assume that $\boldsymbol{\Lambda}$ propagated and updated by Kalman filter from starting state x_0 ,when we get a measerment. We also assume the $\boldsymbol{\Sigma}$ is independent of previous state and is changed with the error quantity from the nominal. It is not propagated from some previous state but from 0.

The importance of the chance constraint is demonstrated by the following figure:

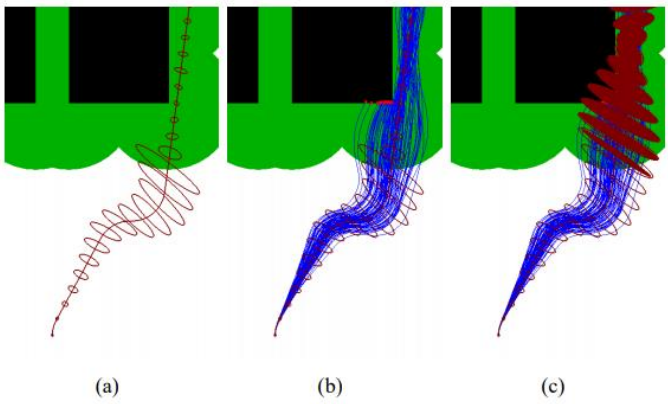

Figure 4

In figure(4) we can see the nominal path generated by the control function with the ellipses of the covariances from not getting an observation in (a), in this case when we reach the green area in which observations can be obtained, we can see the ellipses collapse. In case (b) we can see in blue the estimated paths around the nominal path that are generated using the stochastic motion model. In some cases, we might get a collision with an obstacle due to the uncertainty. In case (c) we can see the updated covariances from the online estimation and the estimation error as we presented above and we can see that the obstacle collisions are considered and expected by the model.

The algorithm:

Before we preset the algorithm, we need to define the data structure with which it works. The data structure the algorithm needs to maintain in real-time. It's important to note that we have work and maintain the sets of data structures in two different spaces – the state space and the belief space.

In the State Space we have: V – Vertices , E – Edges where $v \in V$, $e \in E$, v.x = state

In the Belief Space we have: $v.N$ – Set of belief nodes for each vertex , $n \in v.N$ – set of associated belief nodes $\mathrm{n}.\Sigma$, $\mathrm{n}.\Lambda$, $\mathrm{n}.\mathrm{c}$, $\mathrm{n}.\mathrm{parent}$ – both covariances, cost of node and parent vertex respectively.

Each $e \in E$ contains the output of the connect function

For each node we have the covariances of the local error and the nominal path. Each edge contains the output of the connect function: control low and gain, Simple example we can see in fig (5):

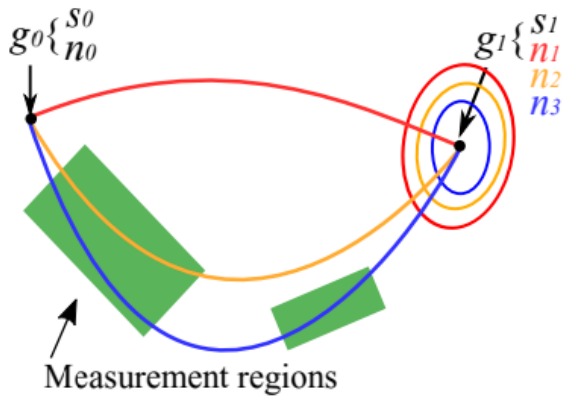

Figure 5

We can see 2 vertices g_0 g_1 that are belong to the state space. in the belief space we have 3 edges with corresponding nodes and their covariances.

additional Functions and data structures:

- Q - Queue of belief nodes
- Functions
  - **Connect**()
  - n_{new} = **Propagate**(e, n_{start})
  - x_{rand} = **Sample**()
  - $v_{nearest}$ = **Nearest**(V, v_{new})
  - V_{InBall} = **Near**(V, v_{new})
  - **AppendBelief**(v, n_{new})

Q: Keeps track of paths that need updating in each iteration of the algorithm.

Propagate: Covariance prediction, cost expectation evaluation and chance constraint checking, if violated returns no belief.

Sample: Returns uniform samples from X free

Nearest: Takes current set of vertices and return the nearest while minimizing some distance function

Near: Return all vertexes within some ball with radius ρ, centered in V_{new} . Similar to extened RRT*.

AppendBelief: Checks if any node dominates the current node and pruning the path if necessary.

Comparing partial paths: Pruning by the AppendBelief function

$$(16)\quad n_a < n_b \Leftrightarrow (n_a.\Sigma < n_b.\Sigma) \wedge (n_a.\Lambda < n_b.\Lambda) \wedge (n_a.c < n_b.c)$$

$$(17)\quad n_a \lesssim n_b \Leftrightarrow \big(n_a.\Sigma < (n_b.\Sigma + \epsilon \mathrm{I})\big) \wedge \big(n_a.\Lambda < (n_b.\Lambda + \epsilon \mathrm{I})\big) \wedge (n_a.c < n_b.c)$$

Equation 16 we can see the partial ordering we are imposing over a couple of nodes in the same vertex, node a dominates node b if covariances and cost function are smaller.

But this can impose a problem which makes the robot prefer to cruse in a loop inside information rich areas, to deal with this problem we can propose a design parameter that lets you control the difference between each corresponding covariance ϵ.

Example of pruning: as you can see in the bottom figure in case (a) the covariance of the red path is smaller than the blue one but overall cost is higher so in this case pruning will not be possible, in (b) we can see that the blue covariance and cost are lower than the red one so we can prune the red path safely.

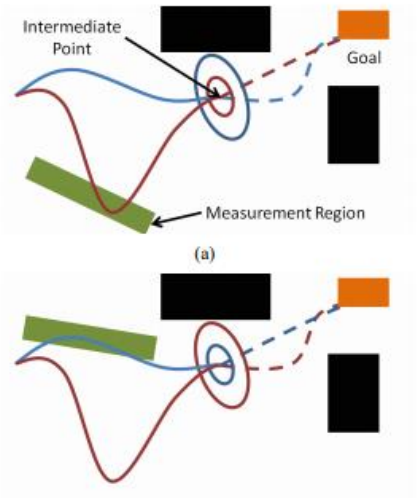

Figure 6

Simplified block diagram of the algorithm RRBT:

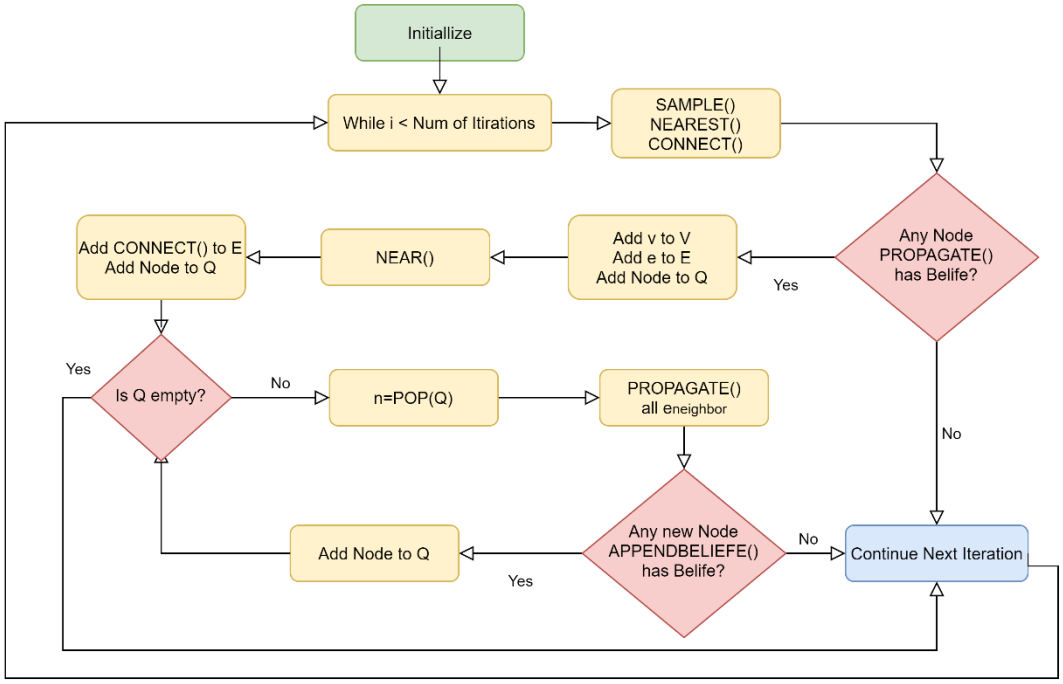

Figure 7

Author's examples:

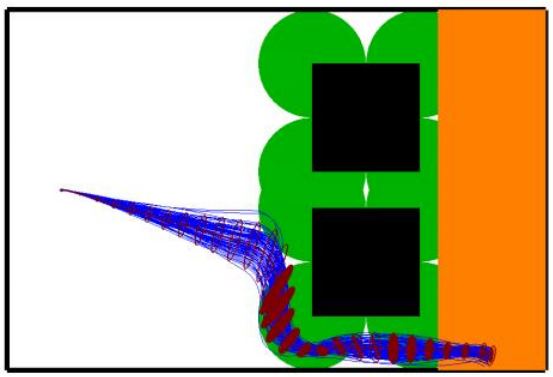

Figure 8

Here we have an implementation of the algorithm on. The algorithm prefers to pass further from the obstacle instead of going through the shortest path to avoid collision, using the updated covariances.

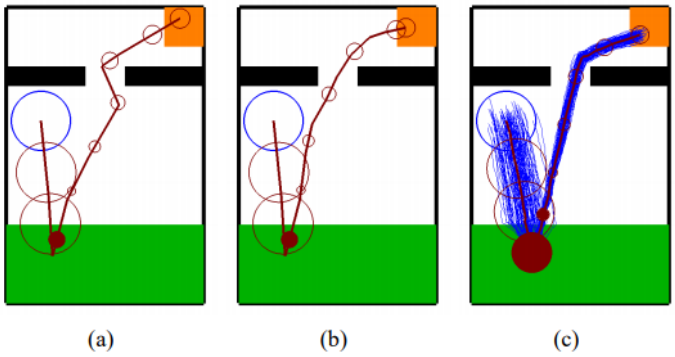

Figure 9

Re-examining the case we showed in figure (1), we can see that after using RRBT in this scenario the system prefers to go to the green area in order to obtain measurements to reduce state uncertainty before going through the narrow pass between obstacles. In blue all possible generated trajectories are collision free and obey the chance-constraint.

Main contribution:

The main results of the paper is the introduction of the RRBT algorithm, it's proof of convergence when taking assumptions and implementing it on an actual system.
The first assumption talks about the connect function – if an output of the connect function between two points in the state space (A,C) is E1 , and point B is part of E1, than the sum of the outputs of (A,B)=E2, (B,C)=E3 must be equal to E1. This is necessary due to the discrete nature of the sampling algorithm, this assumption allows us to interpolate between two close states and control signals and be close enough to the original continuous function.
The second assumption is that for every given point in the state space there is small radius such that the chance constraint condition is not violated – This helps us to keep the graph at finite sample volume to enable convergence.
The third assumption is formulated in a mathematical form – if two beliefs in the state space have the same expectations and different covariances – than the point with the grater covariance has a larger chance to collied with an obstacle.
The fourth assumption is that the cost function is convex , meaning that for any two given belief points in the state space with same expectation and different covariance the expectance of the cost function of the point with the greater covariance is larger. Another assumption is that the obstacles are convex too, and if the cost function is not globally convex it may still be locally convex along the optimal path.
Fifth assumption is that the partial derivatives of the non linear system are exact – this assumption states that the system must be perfectly locally linear and that the LQG properties are constant during the planning and executing phases. This assumption is crucial for us to be able to use the control law in a feedback loop.
After stating the assumptions we can begin talking about the derived lemmas and theorems about the properties of the RRBT algorithm:

Lemma 1 : as the number of iterations of the RRBT approaches to infinity , all of the finite length paths of the free space are represented in the RRBT constructed graph. The proof idea of this lemma relies on assumption 1 and 2 ,and it says that if the obstacles are spaced in such a way

that all of the points are reachable in the belief space so that each ball of a small radius will contain infinite number of graph nodes. Because this holds for every point in the space we will get an infinitely dense connected graph, which will be under the chance-constraint rule.

Lemma 2 : If A and B are two covariance matrices there exist another symmetric positive definite matrix C such that:

$$A^{-1} = (A + B)^{-1} + C$$

the proof is simple by using the general binomical matrix inversion lemma:

$$(A + B)^{-1} = A^{-1} - A^{-1}B(B + BA^{-1}B)^{-1}BA^{-1}$$

this helps us to show that the LQG belief propagation is invariant with respect to inequality in initial beliefs.

Theorem 1 : for two covariance matrices A and B there exist some positive definite matrix C such that A+C=B, this holds for any time step in the interval $[0, \infty]$. This can be proved by the Kalman filter properties and some algebraic manipulation.

Theorem 2 : the second theorem expands the conclusions of theorem 1 to the covariance of the separately propagated belief of the difference between the true state and the nominal, which is used by the algorithm.

Theorem 3 : the third theorem states that for two beliefs n_a and n_b at the same state, p_a and p_b are the corresponding nominal trajectories to the beliefs , ${p^g}_a$ and ${p^g}_b$ are the nominal trajectories to the goal, if $n_a \leq n_b$ than the sum of the expectation of the cost function to the goal and the current const of n_a is also smaller equal to the sum of n_b . this can be proved with assumptions 3 and 4 and theorems 1 and 2. This theorem guarantees that only sub-optimal paths will be pruned and that the cost expectance will be the lowest available.

Implementation:

In this part we will demonstrate simulation results generated by our implementation of the RRBT algorithm. Because of the powerful trait of the RRBT algorithm that is the decupling of the control problem and the path planning problem, we allowed ourselves to make modifications to the algorithm's realization in order to make it less complex in the sense of computability running time.
The first modification we made is we assumed that the motion model's derivatives (or: linearized local motion model) are constant matrices Therefore there is no need to compute the derivatives in every iteration and the LGQ control law becomes simpler inside of the connect function. For that same reason we used a regular KF instead of the extended Kalman filter (EKF). As in the article itself, we made the chance-constraint criteria only to check collision between obstacles and to corresponding covariance matrix drawn in the state space round the nominal point.
We started from implementing a realization of RRT* algorithm and then we inserted additional functions and conditions to correspond to the RRBT algorithm.
Simulation results:
In all of our next figures the blue rectangles represents obstacles, the green and black lines are the connections between sampled points, the black rectangle in the lower right corner represents an area in which the robot can get an observation and the red line is the output chosen valid path by the algorithm. Start point is at (0,0) and the goal is at (50,45). All of the

runs had the same stopping condition – when finding a valid path from start to goal or Maximum number of samples had been reached.

Firstly, for comparison we will show the output of the regular RRT* algorithm on a given 2D map in the state space:

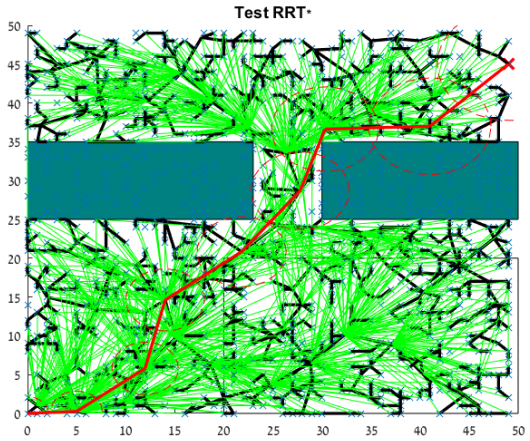

Figure 10

The output of our RRBT algorithm on the same scenario:

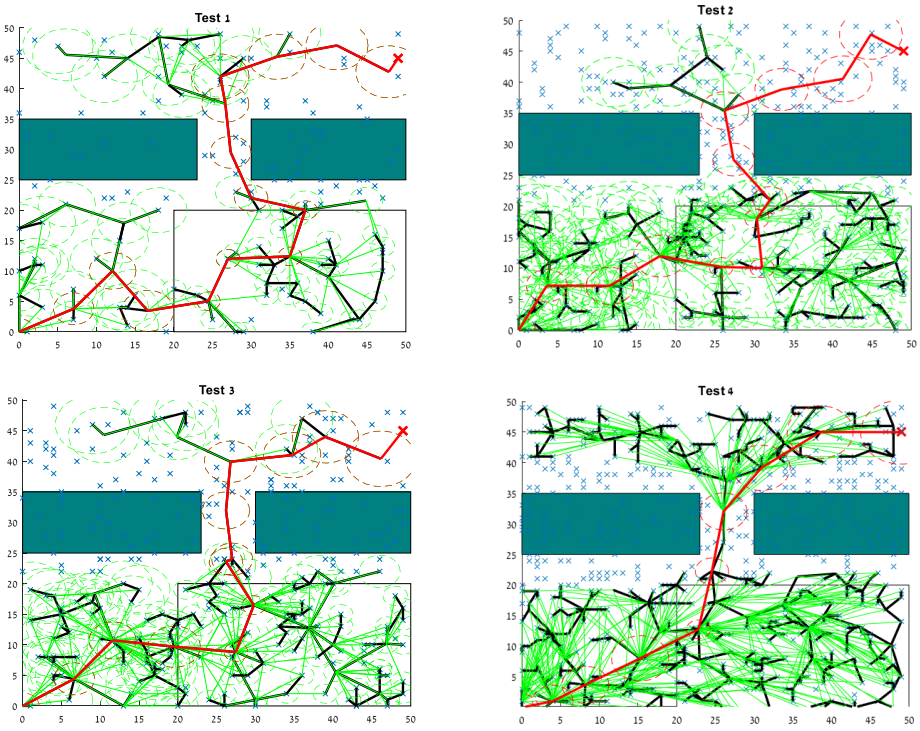

Figure 11

As demonstrated in the RRT* output, the direct path to the goal region without going through measurement area results in violating the chance-constraint conditions, while the output of the RRBT algorithm shows us that only a path through the measurement area is valid and resulting a small enough uncertainty to pass between the obstacles.

As an improvement to the algorithm we suggest a method to terminate the algorithm in the case of passages too narrow for the robot to go through without violating the chance-constraint condition. For this we assume we have corner detecting device such as a sensor that can indicate whether the sampled point which is located in distance smaller than propagating step size located on an obstacle corner (we assume the sensor can determine the direction of the corner). If throughout the run of the algorithm we sampled two corners which are pointing towards each other, we can safely assume that these are two corners of two different obstacles that forms a passage. if the width is narrower than the minimal diameter of the smallest covariance matrix possible, than the algorithm is terminated and returns no solution found.

We found this solution helpful while examining our implementation - when the passage was too narrow the algorithm ran trough the max number of iterations – as you can see in the following figure:

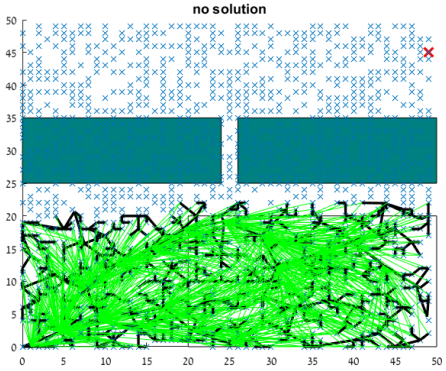

Figure 12

Our small improvement helped us to debug and understand better the inner working of the algorithm.
Our suggested addition may be used to iterate over some chance-constraint sizes (allowed probability of collision) and find the minimum probability of collision possible to reach the goal. Also for further research we might suggest such implementation that prunes out entire areas in which passage is not possible, by reducing the size of sample free space we can reduce the calculation time of the algorithm.

*** All scripts and function are attached separately.

Discussion and Conclusions:

In this report we talked about the sampling based algorithm proposed in the article Rapidly-exploring Random Belief Trees for Motion Planning Under Uncertainty by Adam Bry Nichols Roy. The article provided a background on the current state of the art algorithms that were available at the time and showed the "gap" of dealing with uncertainty in nonlinear systems. They introduced their proposed algorithm RRBT and provided for examples and partial proofs for its capabilities, advantages and limitations. Finally they showed some experimental results that demonstrated the algorithm.
We implemented a version of this algorithm by taking some assumption and essentially expanding over an RRT* algorithm, and we also suggested a minor modification to terminate the algorithm run when there is no chance to find valid path.
This paper set a foundation to many other papers that expanded upon it such as "anytime-RRBT" or "FRRBT – feedback RRBT".

Few issues that need to be examined in further research:

- Excessive number of belief nodes are generated in each vertex – even with the pruning feature.
- Unsuitable in Rea-time application
- Local linearization is not suitable for all non-liner systems
- Result of Sub-optimal path is common due to bias towards exploring information rich areas.
- Not suitable for dynamic environments.
- Not considering unavailability of a valid passage (see our proposed solution for this problem)

Sources:

- ❑ H. Young J.Lim. Anytime RRBT for Handling Uncertainty and Dynamic Objects 2016
- ❑ V,Pilania ,L.Gupta A Localization Aware Sampling Strategy for Motion Planning under Uncertainty 2015
- ❑ M. Achtelik, S.Weiss Path Planning for Motion Dependent State Estimation on Micro Aerial Vehicles 2013
- ❑ S. Karaman and E. Frazzoli. Incremental sampling-based optimal motion planning. In Robotics: Science and Systems, 2010
- ❑ A.Bry Control, Estimation, and Planning Algorithms for Aggressive Flight using Onboard Sensing 2012
- ❑ E. Frazzoli. Robust Hybrid Control for Autonomous Vehicle Motion Planning. Department of aeronautics and astronautics, Massachusetts Institute of Technology, Cambridge, MA, June 2001.
- ❑ S. M. LaValle. Planning Algorithms. Cambridge University Press, Cambridge, U.K., 2006. Available at http://planning.cs.uiuc.edu/.
- ❑ D. Mellinger, N. Michael, and V. Kumar. Trajectory generation and control for precise aggressive maneuvers with quadrotors. In Int. Symposium on Experimental Robotics, 2010
- ❑ A. Bry and N. Roy. Exact belief state computation for piecewise LQG planning. Technical report, Massachusetts Institute of Technology, 2010.