Bidirectional Sampling Based Search
Without Two Point Boundary Value Solution

Sharan Nayak and Michael W. Otte

Abstract—Bidirectional path and motion planning approaches decrease planning time, on average, compared to their unidirectional counterparts. In the context of single-query feasible motion planning, using bidirectional search to find a continuous motion plan requires an explicit connection between the forward search tree and the reverse search tree. Such a tree-tree connection requires solving a two-point Boundary Value Problem (BVP). However, two-point BVP solution can be difficult or impossible to calculate for many types of vehicles (using numerical methods to find a solution, such as shooting approaches may be computationally expensive and is sometimes numerically unstable). To overcome this challenge, we present a generalized bidirectional search algorithm that does not require solving two-point BVP. Instead of connecting the two trees directly, our algorithm uses the cost information of the reverse tree as a guiding heuristic for forward search. This enables the forward search to quickly converge to a full feasible solution without an explicit tree-tree connection and without the solution to a two-point BVP. We run multiple software simulations in different environments and using dynamics of different vehicles along with real-world hardware experiments to show that our approach performs very close or better than existing state of the art approaches in terms of quickly converging to an initial feasible solution.

Index Terms—Motion and Path Planning, Autonomous agents and Dynamics

I. INTRODUCTION

AMPLING based motion planning approaches have become popular in robotics due to their simplicity and rapid exploration of high-dimensional state spaces. They involve randomly sampling collision-free feasible states of a robot in the configuration space and moving between them in a way that respects a robot’s kinematics and dynamics. Quick feasible planning [1] and long-term asymptotically optimal motion planning [2][3][4] are two problems in single-query motion planning that have garnered interest in the motion planning community. Quick feasible planning allows a robot to quickly find an initial feasible solution and immediately start making progress towards the goal while long-term asymptotically optimal motion planning allows for calculating improved quality solutions over time. In this paper, we focus on the quick feasible motion planning problem.

Single-query searches for feasible motion planning can be divided into unidirectional and bidirectional searches (Fig. 1). Unidirectional search has a single tree built from the start state outwards to the end state whereas bidirectional searches have two search trees expanding outwards from the start and end configurations which are connected when “close” to each other. Bidirectional searches have been popular due to their improved performance in exploring higher dimensional search spaces and spaces with narrow passages [6]. However many existing bidirectional search approaches [6][7][8] have
focused on solving the planning problem when a steering function is available i.e. when the two point boundary value problem (BVP) is solvable \([5,9]\) (Fig. 2). However it is difficult to come up with closed-form solutions for many kinodynamic robots especially with non-holonomic constraints. Moreover, even if the closed form solutions exist, they are usually solved using numerical methods like shooting approaches \([10]\), which can often be computationally expensive when faced with real-world planning time constraints. Furthermore, it is not possible to include every maneuver that solves 2-point BVP in pre-computed maneuver libraries \([11,12]\) which have been used to speed up executions of motion planners. This is due to memory constraints involved in storing the maneuvers. On the other end, existing bidirectional approaches that do not require solving the two point-BVP for building the trees \([13]\) suffer from discontinuities when connecting them and additional methods like curve-smoothing \([14]\) have been employed to remove the discontinuities which are non-trivial.

The main contribution of this paper is a new bidirectional single-query sampling based search approach that returns a continuous motion plan without solving a two-point BVP. We call this method Generalized Bidirectional RRT (GBRRRT). Our method (Fig. 3) differs from the RRT-Bidirectional \([13]\) in that it does not try to connect the two trees when they are close, rather it uses the heuristic information provided by the reverse search tree to make the forward tree quickly grow towards the end configuration. We prove that the provided version of GBRRT in this letter is probabilistically complete. We run multiple experiments in simulation using unicycle and quadrotor dynamics over different test environments to show the effectiveness of our proposed algorithm. We also run real-vehicle experiments using a quadrotor to show the improvement in performance over existing state of the art methods in large obstacle environments.

The rest of the paper is organized as follows. Section II provides a discussion on related work. Section III provides the formal problem definition for our work. Section IV provides a discussion of our proposed algorithms. Section V provides the analysis for our algorithm for probabilistic completeness. Section VI explains the setup used for running experiments. Section VII provides the result of the experiments. Section VIII concludes by summarizing the contributions and main results.

II. RELATED WORK

There have been a variety of bidirectional-search sampling algorithms proposed in literature which we now survey. One of the seminal work on bidirectional sampling based search is RRT-Connect \([13]\) which uses a greedy heuristic to provide fast convergence towards an initial feasible solution. Jordan et al. \([6]\) develop a bidirectional variant of RRT* \([2]\) that provides asymptotic optimality utilizing various heuristics and procedures to improve the convergence rate. Tahir et al. \([15]\) and Xinyu et al. \([16]\) use artificial potential fields \([17]\) to generate intelligent samples using gradient descent methods to improve speed of convergence of bidirectional RRT*. Devaurs et al. \([7]\) expand Transition-RRT (T-RRT) to create a bidirectional T-RRT to achieve faster convergence and better quality paths over configuration spaces where the cost functions are defined. Starek et al. \([8]\) extend FMT* to develop the Bidirectional Fast Marching trees (BFMT*) which they show has better convergence rates for high quality paths than many existing unidirectional planners. Although these algorithms although provide fast convergence rates, they cannot be used if the 2-point BVP cannot be solved (or in practice when doing so is computationally expensive). The existing bidirectional algorithms \([13,14]\) that do not require solving 2-point BVP, instead require extra computation to connect the discontinuity that exists between the two trees, e.g., using methods like point perturbation \([15]\) or Bézier curves \([14]\). Our approach avoids the tree-tree connection problem entirely, by using the heuristic provided by the reverse search tree to quickly move towards the goal.

The solution generation to a 2-point BVP usually involves solving a differential equation constrained to the given start and end boundary conditions. This is non-trivial for robots with complex dynamics. Hence researchers have looked into solving the motion planning problem either by generating approximations to the two-point BVP \([18]\] using shooting approaches \([10]\) or by simplifying the dynamics by linearization \([19]\) and then solving 2-point BVP. More recently, Li et al. \([20]\) propose the Stable-Sparse Trees (SST and SST*) who show that asymptotic optimality can be achieved by only performing random forward propagation without solving 2-point BVP. Moreover, SST prunes the dominated nodes always maintaining a parse set of nodes in the search tree. Although SST provides asymptotically optimal paths, it is slow in convergence in practice towards high quality paths due to the use of random controls \([21]\). Informed SST (iSST) \([22]\) and Dominance-Informed Region Trees (DIRT) \([23]\) provide improved rates of convergence towards high quality trajectories by using heuristics while still maintaining asymptotic optimality. A common theme in random-propagation based algorithms is that there are unidirectional based. However, in expectation, bidirectional searches provide improved efficiency and faster convergence rates \([6,5]\) than unidirectional searches. Therefore we present a new type of bidirectional search that does not require solving 2-point BVP.

A well chosen heuristic can often improve search efficiency. A variety of heuristics have previously been used in the context of sampling based motion planning. A common approach is sampling the goal node with a certain probability, also known as goal biasing \([13]\). Akgun et al. \([24]\) use sampling heuristics to sample more nodes near a found path to help the current path quickly improve towards a more optimal one. Urmson et al. \([25]\) propose the heurstically guided RRT (hRRT) where a quality measure is used to balance exploration vs exploitation in selecting the next node to expand which leads to low cost solutions. DIRT \([23]\) and iSST \([22]\) use user-defined heuristics to provide high rates of convergence towards optimal solution. Reaping the benefits of a heuristic, in our proposed algorithm, the Cost to End (CTE) heuristic provided by the reverse search tree is used to improve our rate of convergence towards an initial feasible solution.
III. PROBLEM DEFINITION

Let \( \mathcal{X} \) be the \( D \)-dimensional configuration space of a robot. Let the obstacle space \( \mathcal{X}_{\text{obs}} \) be an open subset of \( \mathcal{X} \). \( \mathcal{X}_{\text{obs}} \) is the set of all configurations where the robot is in collision with the workspace obstacles. The free space \( \mathcal{X}_{\text{free}} \) is defined as \( \mathcal{X}_{\text{free}} = \mathcal{X} \setminus \mathcal{X}_{\text{obs}} \) is the closed subset of \( \mathcal{X} \). Let the start configuration be denoted by \( x_{\text{start}} \) and the end region be \( x_{\text{end}} \) with \( x_{\text{start}} \in \mathcal{X}_{\text{free}} \) and \( x_{\text{end}} \subset \mathcal{X}_{\text{free}} \). Let \( x_i, x_{i+1} \in \mathcal{X}_{\text{free}} \) and let \( \alpha_{i}^{+1} \) be a function that satisfies \( \alpha_{i}^{+1} : [0, 1] \to \mathcal{X}_{\text{free}} \) such that \( \alpha_{i}^{+1}(0) = x_{i} \) and \( \alpha_{i}^{+1}(1) = x_{i+1} \). \( \alpha_{i} \) is a valid maneuver that solves the two-point BVP for configurations \( x_i \) and \( x_{i+1} \) given the robot’s kinematic and dynamic constraints. Let \( \beta \) be a feasible path defined as ordered set of valid maneuvers \( \{\alpha_0^1, \alpha_2^2, \ldots, \alpha_{n-1}^n\} \).

**Problem:** Single-Query feasible motion planning

Given \( \mathcal{X}_{\text{free}}, x_{\text{start}} \) and \( x_{\text{end}} \), find \( \beta \) such that \( \alpha_{0}^1(0) = x_{\text{start}} \) and \( \alpha_{n-1}^n(1) = x_{\text{end}} \).

IV. ALGORITHM DESCRIPTION

The main idea behind our proposed GBRRT algorithm is to use heuristic information from reverse search tree to guide the advancement of the forward search tree towards the goal. The search begins by using the forward and reverse search tree to explore different parts of the configuration space starting from the start and end query points, respectively. The exploration search in both the trees basically correspond to search conducted by classic RRT [13] algorithm. Once the two trees encounter each other, the forward tree combines ongoing exploration with an exploitation strategy that leverages the CTE values stored in the reverse tree to direct the search. This causes the forward search tree to quickly move away from the obstacles and towards the end goal. In the next subsection (Subsection IV-A), we briefly explain the classic RRT [13] algorithm and then how to extend it to the proposed GBRRT algorithm (Subsection IV-B).

A. Rapidly Exploring Random-Tree (RRT)

The inputs to the RRT [13] (Algorithm 1) are the start node \( v_{\text{start}} \) and end node \( v_{\text{end}} \). The search tree \( G_{\text{for}} \) is initialized using the node set \( \mathcal{V} \) (initialized using \( v_{\text{start}} \)) and edge set \( \mathcal{E} \) (line 1). Each iteration of RRT executes the function \text{RRTSingleIteration} (Algorithm 1) which generates a new potential edge \( E_{\text{new}} \) for addition to the search tree using the getEdgeExtend (Algorithm 1) function. \( E_{\text{new}} \) is created using the chain of the following familiar operations - RandomConfig, NearestNeighbor and Extend as given in [13]. We will call this the “classic” RRT way of edge generation in the coming sections. \( E_{\text{new}} \) is checked for collision using CollisionCheck and gets added to the search tree if no collision exists (line 5 in Algorithm 2). The new added node \( v_{\text{new}} \) is used for checking if the end region is reached using function endReached (line 4 in Algorithm 1).

Algorithm 1 returns the output path generated using function Path if the end is reached (line 5) or otherwise continues to the next iteration.

B. Generalized Bidirectional RRT (GBRRRT)

GBRRRT (Algorithm 2) accepts the same inputs as RRT (Algorithm 1) with two additional inputs: \( \delta_{\text{nr}} \) (neighbor radius) and \( \delta_{\text{hr}} \) (heuristic radius). Each of the new parameters will be explained in the coming sections. The additional reverse search tree \( G_{\text{rev}} \) needed for bidirectional search is initialized using the node set \( \mathcal{V}_{\text{rev}} \) (initialized using \( v_{\text{end}} \)) and edge set \( \mathcal{E}_{\text{rev}} \) (line 2). We use a priority queue \( Q \) to maintain a list of potential reverse tree nodes to guide the expansion of the forward search tree (line 3). In each iteration, we expand the reverse and forward search trees as shown in line 5 and 6-15 respectively. The expansion of the reverse tree occurs in the same way as classic RRT tree expansion (Algorithm 6). The expansion of the forward tree is different from the reverse tree in that it uses a combination of classic (random) search and heuristic guided exploitation. The heuristic exploitation is based on using the CTE heuristic derived from the expanding reverse tree to guide the forward tree growth towards the goal.

The exploitation ratio \( q_k \) (line 6) determines the probability of whether the forward search tree performs exploitation (heuristic expansion) over exploration (random) in the current iteration. The user-defined function getExploitationRatio is used to calculate \( q_k \). In our implementation, we choose getExploitationRatio to return a value calculated using the expression \( R_{\text{exp}} e^{-c_k} \) where \( R_{\text{exp}} \) is the initial exploitation ratio, \( k \) is the iteration number and \( c \) is a large positive constant. It will be shown in Section V that using this expression causes our algorithm to be probabilistic complete. The algorithm uses getEdge (Algorithm 3) in line 7 to get the new potential edge \( E_{\text{new}} \) to add to the forward search
vertices set \( V_{for, near} \) to \( v_{next} \) in \( G_{for} \) and selects the best vertex \( v_{near} \) which has the minimum cost \( g(v) + d(v_{next}, v) + h(v_{next}) \) where \( g(v) \) is the Cost From Start (CFS) for node \( v \), \( d(v_{next}, v) \) is the euclidean distance between \( v_{next} \) and \( v \) and \( h(v_{next}) \) is the CTE for node \( v_{next} \). The algorithm next uses the extend operator (line 5) to generate the next potential edge \( (v_{near}, v_{new}) \) to add to the tree. If \( v_{next} \) is NULL in line 7 which implies \( Q \) is empty, the algorithm exits returning NULL (line 10).

Algorithm 4: getEdgeHeuristic(\( G_{for}, Q, \delta_{hr} \))

\[
\begin{align*}
1 & v_{next} \leftarrow \text{Pop}(Q) \\
2 & \text{if } v_{next} \not= \text{NULL then} \\
3 & \quad V_{for, near} \leftarrow \text{NearestVertices}(v_{next}, G_{for}, \delta_{hr}) \\
4 & \quad v_{near} \leftarrow \min_{v \in V_{for, near}} (g(v) + d(v_{next}, v) + h(v_{next})) \\
5 & \quad v_{new} \leftarrow \text{Extend}(v_{near}, v_{next}) \\
6 & \quad \text{if not } v_{near}.\text{Neighbor}(G_{for}, v_{new}) \text{ then} \\
7 & \quad \quad \text{return } (v_{near}, v_{new}) \\
8 & \text{return } \text{NULL}
\end{align*}
\]

Update of Priority Queue \( Q \): The addition of the reverse tree nodes to the priority Queue \( Q \) (Fig. 3) is performed by Algorithm 5. The algorithm first determines the nearest vertices set \( V_{rev, near} \) to newly added forward tree node \( v_{new} \) within a ball of radius \( \delta_{nr} \) (line 1). If \( V_{rev, near} \neq \emptyset \), then for each vertex \( v \in V \), we use the CTE heuristic \( h(v) \) to update \( Q \). The condition of \( V_{near} = \emptyset \) is satisfied at the start of the search when the forward tree has not encountered the reverse search tree yet.

Algorithm 5: updatePriorityQueue(\( G_{rev}, Q, \delta_{nr}, v_{new} \))

\[
\begin{align*}
1 & V_{rev, near} \leftarrow \text{NearestVertices}(G_{rev}, v_{new}, \delta_{nr}) \\
2 & \text{if } V_{rev, near} \neq \emptyset \text{ then} \\
3 & \quad \text{for } v \text{ in } V_{rev, near} \text{ do} \\
4 & \quad \quad Q.\text{update}(v, h(v))
\end{align*}
\]

C. Selection of parameters

There are two main parameters that affect the performance of GBRT: the neighbor radius \( \delta_{nr} \), and the heuristic radius \( \delta_{hr} \). The neighbor radius \( \delta_{nr} \) affects the number of potential reverse nodes that are added to the priority queue. A low \( \delta_{nr} \) value decreases the likelihood that reverse nodes are considered for heuristic expansion. A high value may increase the likelihood that reverse nodes will be blocked by obstacles. In our experiments, we set \( \delta_{nr} \) to be equal or slightly greater than the twice the length of the largest maneuver. The heuristic radius \( \delta_{hr} \) should be set large enough there are enough forward tree nodes considered for connection. We use \( \delta_{hr} = \delta_{nr}/2 \) in our experiments.

As mentioned in Subsection 4.B, we use the expression \( R_{exp}e^{-h/c} \) to calculate \( q_{k} \). \( R_{exp} \) dictates the classic trade-off between the exploration and exploitation at the search. Fig. 5 shows that computation time \( t_c \) of the initial feasible solution decreases as \( R_{exp} \) approaches 1. We choose \( R_{exp} \) to be 0.8 in our experiments to prioritize more exploitation at the start of search to quickly get to a feasible solution. The decay constant \( c \) affects the rate at which the exploitation ratio decreases to zero. In our experiments, we set \( c \) to 10000.
In this section, we provide proofs for two main results. The first is a proof describing a property of exploration vs. exploitation function that is sufficient to guarantee probabilistic completeness. The second is that using $q_k = R_{exp} e^{-k/c}$ leads to probabilistic completeness. In terms of preliminaries, let $p_k$ be the probability of exploration at $k$-th iteration defined as $p_k = 1 - q_k$. Let $N_{total}$ be the total number of iterations executed in the algorithm and let $N_{explore}$ be the number of explorations done over time. To prove the main results, we first prove the probabilistic completeness of GBRRT assuming $p_k$ has the property that $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$ (Theorem 1), then we show that this property holds for when $p_k = p$ for all $k$ (where $p$ is constant) (Lemma1) as well when $p_k = 1 - R_{exp} e^{-k/c}$ (Lemma 2). It follows that using $p_k = 1 - R_{exp} e^{-k/c}$ in GBRRT makes it probabilistically complete (Theorem 2).

**Theorem 1.** If $p_k$ evolves such that $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$, then GBRRT is probabilistically complete.

**Proof.** To prove this theorem, we will leverage RRT*’s neighborhood proofs. Let us assume a RRT* tree that contains only the nodes added from those iterations that performed exploration in GBRRT algorithm such that the neighborhood radius in RRT* is less than the neighborhood radius $\delta_{nr}$ of GBRRT. Our GBRRT tree will contain a subgraph containing only the nodes of this RRT* tree. We know as $N_{total} \to \infty$, RRT* is probabilistically complete [2] and hence due to the subgraph argument, GBRRT is probabilistically complete. In proving this theorem, we make the assumption that $\delta_{nr}$ is greater than or equal to the neighbor radius of RRT*. (NOTE: even though the node set of RRT* is a subset of GBRRT, the edge set may be different but the tree is still connected which is all that is required for the proof).

**Lemma 1.** If $p_k = p$ for all $k$ where $p$ is a constant and $p > 0$, then $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$.

**Proof.** Let $x_k$ be a binary random variable that assumes a value 1 if exploration is performed in the $k$-th iteration and 0 otherwise. We know from the strong law of large numbers that with probability 1, $\sum_{k=1}^{\infty} x_k / N_{total} \to p$ as $n \to \infty$. Thus (recall $p > 0$), $\sum_{k=1}^{\infty} x_k \to \infty$ as $N_{total} \to \infty$. Using this, we conclude $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$.

**Lemma 2.** if $p_k = 1 - R_{exp} e^{-k/c}$, then $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$.

**Proof.** From Lemma 1 if $p_k$ is constant, then $P(N \to \infty) = 1$ as $k \to \infty$. If we define $p_k = 1 - R_{exp} e^{-k/c}$, then $p_k$ increases after each iteration. Thus the number of iterations where exploration is performed when $p_k$ is constant acts a lower bound when using the given $p_k$. Thus for the given $p_k$, $P(N_{explore} \to \infty) = 1$ as $N_{total} \to \infty$.

**Theorem 2.** The version of GBRRT we use in our experiments is probabilistically complete.

**Proof.** From Theorem 1 and Lemma 2 it follows that our version of GBRRT which uses $p_k = 1 - R_{exp} e^{-k/c}$ is probabilistically complete.

**VI. EXPERIMENT SETUP**

We run multiple software simulations and hardware experiments to test performance of our proposed bidirectional search algorithm. We choose computation time of the initial solution ($t_c$), path length of the initial solution ($l_p$) and flight time ($t_f$) (only hardware experiments) as the performance metrics.

- **A. Simulation experiments**

The simulation experiments are conducted in four different environments (Fig. 7) which includes an environment with randomly dispersed obstacles (DISPERSED), an environment with large obstacles (LARGE), a maze environment and an environment with a narrow passage (NARROW). The LARGE, MAZE, NARROW environment are fixed in experiments but the DISPERSED environment has the number of obstacles varied in the range $\{0, 5, 15, 25, 35\}$. The environments are set to size 100m $\times$ 100m with the distance between start and end locations set to a value $\geq 70$m. We run 50 scenarios for each environment with each scenario having 3 trials (hence
total of $50 \times 3 = 150$ experiments). A scenario in the LARGE, MAZE and NARROW environment is defined as a random selection of start and end location (ensuring distance $\geq 70$) whereas a scenario in the DISPERSED environment corresponds to random selection of start and end location and random positions of different sized rectangular and circular obstacles.

We run the experiments using two vehicles – a 2D quadrotor and an unicycle. The quadrotor is controlled using feedback linearization \textsuperscript{26} and the unicycle is controlled using a time-varying non-linear controller \textsuperscript{27}. We use these controllers to generate maneuver/trajectory libraries (Fig. \textsuperscript{6}) for these two vehicles with the start and end velocities of each maneuver being zero. The zero start and end velocities help reduce the number of maneuvers in our considered maneuver libraries but this assumption should not affect the generality of the experiments. The maneuver library for the quadrotor is generated by providing final positions 1 to 5 metres distance away from the initial position (origin) in $10^\circ$increments. The maneuver library for the unicycle is generated by varying the peak linear velocities from 1 to 5 m/s and the angular velocities varying from 0 to $\pi/2$ rad/s in $\pi/18$ rad/s increments in all 4 quadrants.

We compare GBRRT with four other state of the art single-query algorithms - RRT\textsuperscript{11}, SST\textsuperscript{20}, iSST\textsuperscript{22} and DIRT \textsuperscript{23}. We only include those algorithms for comparison that do not require a two-point BVP solver in order to assure a fair comparison. iSST and DIRT are heuristic based algorithms and we use PRM \textsuperscript{28} to build a roadmap (in each experiment) to provide the heuristic before actually running them. We set the GBRRT parameters $\delta_{nr} = 10$ and $\delta_{hr} = \delta_{nr}/2 = 5$ and SST parameters $\delta_{BN} = 3$ and $\delta_{S} = 1.5$. Both SST and DIRT use blossom-like propagation procedure \textsuperscript{29} and we set the number of maneuvers $M$ in the blossom to 10. The end region radius $\delta_{end}$ is set to 5. The algorithms are implemented in Python 2.7 and the experiments are performed on a system with Intel i7-7700 4-core CPU with 32GB RAM.

\textbf{B. Hardware experiments}

The hardware experiments are performed in an obstacle course of size $5m \times 4m$ with start and end locations set at (0.2, 2.5) and (4.5, 2.5) respectively. The obstacle course is traversed using a modified Bebop 2 quadrotor \textsuperscript{30} with the motion of the quadrotor limited to a horizontal plane at a fixed height. The quadrotor is equipped with a Intel UP board (Fig. \textsuperscript{8}) having Intel Atom x5-z8350 4-core, 1.44GHz CPU with 4GB RAM for running the algorithms. We use a
Proportional-Derivative (PD) controller for position tracking and a Vicon motion capture system for state information. The obstacles in the course are modelled using circular and rectangular areas with radius and length increased by 20 cm to account for vehicle size and drift. As the obstacle course contains large obstacles, we take the two best performing algorithms (GBRRT and RRT) in the LARGE obstacle environment in simulation experiments and perform the comparison. We set GBRRT parameters $\delta_{nr} = 1$ and $\delta_{hr} = \delta_{nr}/2 = 0.5$. $\delta_{end}$ is set to 0.3.

**VII. RESULTS**

The initial feasible paths generated by GBRRT for quadrotor and unicycle in the different test environments for one run are shown in Fig. 7. The graphs for the computation time vs. obstacles number and path length vs. obstacles number for the DISPERSED environment is shown in Fig. 9 and the results for other environments are listed in Table I. The results indicate that GBRRT perform better than all algorithms for all obstacle number cases for both the vehicles in terms of computation time ($t_c$) for the DISPERSED environment. A similar conclusion can be made for the LARGE (for both vehicles), MAZE (quadrotor) and NARROW (quadrotor) environments. iSST performs better than all tested algorithms in $t_c$ for the unicycle in MAZE and NARROW environment. We believe this can be attributed to the heuristic (generated by PRM roadmap) in guiding the unicycle which has shorter and more curved maneuvers. iSST and DIRT do not perform well in the DISPERSED and LARGE environment for $t_c$ because generating the roadmap for the heuristic takes a significant chunk of time because of the number of obstacles present in these environments which is added to their overall $t_c$ to generate the solution. In terms of path length ($l_p$), we find that GBRRT performs better than all algorithms (except for DIRT in DISPERSED environment for quadrotor vehicle) in all environments. We believe this happens because GBBRT uses more direct maneuvers when moving towards the end goal when perform the exploitation search. Although iSST and DIRT use heuristics to move towards the goal, their solution quality is found be less than GBRRT. We believe this can be attributed to the randomness involved in the selecting the maneuvers for the blossom propagation at each expanded node.

The initial feasible path generated by GBRRT in one of the trials of the hardware experiments with the actual path (using Vicon) is displayed in Fig. 8 and results from 5 trials of hardware experiments is shown in Table II. We find that GBRRT on average performs better than RRT in all performance metrics. A similar conclusion was obtained in our simulations and thus our hardware experiments corroborate the findings of our simulation experiments.

**TABLE I: Performance metrics for all algorithms in different environments**

| Vehicle | Env. | Alg. | GBRRT | RRT | SST | iSST | DIRT |
|---------|------|------|-------|-----|-----|------|------|
| Quadrotor | LARGE | Metric | $t_c$ (s) | $l_p$ (m) | $t_c$ (s) | $l_p$ (m) | $t_c$ (s) | $l_p$ (m) | $t_c$ (s) | $l_p$ (m) |
| Mean | 0.36 | 115.03 | 0.61 | 122.32 | 2.89 | 149.08 | 0.89 | 122.55 | 0.89 | 116.72 |
| Std. | 0.03 | 3.25 | 0.04 | 3.13 | 0.2 | 4.13 | 0.01 | 3.18 | 0.01 | 3.07 |
| MAZE | Mean | 1.37 | 150.43 | 2.38 | 158.83 | 13.84 | 204.84 | 1.63 | 164.16 | 1.79 | 160.12 |
| Std. | 0.08 | 5.48 | 0.24 | 5.8 | 1.56 | 7.8 | 0.03 | 5.93 | 0.05 | 5.66 |
| NARROW | Mean | 0.53 | 128.86 | 0.55 | 134.62 | 1.78 | 166.23 | 0.77 | 143.84 | 0.76 | 134.71 |
| Std. | 0.05 | 2.55 | 0.05 | 2.7 | 0.1 | 4.04 | 0.01 | 3.04 | 0.01 | 2.82 |
| Unicycle | LARGE | Mean | 0.89 | 122.21 | 1.51 | 132.17 | 5.48 | 155.58 | 1.31 | 144.77 | 4.28 | 128.57 |
| Std. | 0.06 | 3.45 | 0.12 | 3.55 | 0.46 | 4.37 | 0.02 | 4.02 | 0.47 | 3.32 |
| MAZE | Mean | 3.44 | 155.74 | 5.37 | 169.7 | 24.9 | 209.88 | 2.26 | 197.49 | 22.04 | 175.41 |
| Std. | 0.19 | 5.7 | 0.51 | 6.15 | 2.4 | 7.75 | 0.04 | 6.75 | 5.53 | 6.32 |
| NARROW | Mean | 2.02 | 136.78 | 2.21 | 150.08 | 3.71 | 174.09 | 1.22 | 165.18 | 5.88 | 149.49 |
| Std. | 0.18 | 2.83 | 0.25 | 3.11 | 0.22 | 3.56 | 0.02 | 3.37 | 1.37 | 3.07 |

**TABLE II: Mean and standard deviation of performance metrics for 5 trials of hardware experiments**

| Algorithm | GBRRT | RRT |
|-----------|-------|-----|
| Metric | $t_c$ (s) | $l_p$ (m) | $t_c$ (s) | $l_p$ (m) |
| Mean | 1.34 | 5.59 | 23.55 | 1.47 | 3.59 |
| Std. | 0.56 | 0.22 | 32.36 | 0.42 | 0.65 |

FIG. 9: Figures show computation time vs. number of obstacles and path length vs. number of obstacles for Quadrotor (Left) and Unicycle (Right) respectively for the DISPERSED environment. Each point is a mean of values from 50 scenarios with each scenario having 3 trials.
VIII. CONCLUSION
We present Generalized Bidirectional RRT (GBRRT), a new bidirectional sampling based algorithm that does not require a two-point BVP solver and produces a continuous motion plan for solving the initial feasible motion planning problem. This algorithm instead of trying to connect the forward and reverse trees, makes the forward tree use the Cost To End (CTE) heuristic provided by reverse search tree to quickly converge to an initial feasible solution. We provide proofs to outline conditions under which GBRRT is probabilistically complete. We execute multiple simulation experiments in different environments and using different vehicles along with hardware experiments to show the effectiveness of our proposed algorithm using computation time ($t_c$), path length ($l_p$) and flight time ($t_f$) (only hardware) as the performance metrics.

Our results show that GBRRT performs better than existing state of the algorithms in DISPERSED and LARGE environments (both quadrotor and unicycle) and MAZE and NARROW environments (quadrotor) and the real obstacle course. GBRRT does better than all algorithms in the path length metric which can be attributed to the direct maneuvers (towards the goal) performed by GBRRT during exploitation search. This shows the power of performing a bidirectional search and using a heuristic in increasing the rate of convergence and producing good quality feasible solutions.

ACKNOWLEDGMENT
We like to thank Nitin Sanket and Chahat Deep Singh from the Perception and Robotics Group (PRG), UMD for allowing us to use the quadrotor system for our hardware experiments.

REFERENCES
[1] S. M. LaValle, “Rapidly-exploring random trees: A new tool for path planning,” 1998.
[2] S. Karaman and E. Frazzoli, “Sampling-based algorithms for optimal motion planning,” The international journal of robotics research, vol. 30, no. 7, pp. 846–894, 2011.
[3] O. Arslan and P. Tsiontsos, “Use of relaxation methods in sampling-based algorithms for optimal motion planning,” in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 2421–2428.
[4] M. Ote and E. Frazzoli, “Rrtx: Asymptotically optimal single-query sampling-based motion planning with quick replanning,” The International Journal of Robotics Research, vol. 35, no. 7, pp. 797–822, 2016.
[5] S. M. LaValle, Planning algorithms. Cambridge university press, 2006.
[6] M. Jordan and A. Perez, “Optimal bidirectional rapidly-exploring random trees,” 2013.
[7] D. Devaurs, T. Siméon, and J. Cortés, “Enhancing the transition-based rrt to deal with complex cost spaces,” in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 4120–4125.
[8] J. Starek, E. Schmerling, L. Janson, and M. Pavone, “Bidirectional fast marching trees: An optimal sampling-based algorithm for bidirectional motion planning,” in Workshop on Algorithmic Foundations of Robotics, 2014.
[9] H. B. Keller, Numerical methods for two-point boundary-value problems. Courier Dover Publications, 2018.
[10] M. R. Osborne, “On shooting methods for boundary value problems,” Journal of mathematical analysis and applications, vol. 27, no. 2, pp. 417–433, 1969.
[11] M. S. Branicky, R. A. Knepper, and J. J. Kuffner, “Path and trajectory diversity: Theory and algorithms,” in 2008 IEEE International Conference on Robotics and Automation. IEEE, 2008, pp. 1359–1364.
[12] J. Go, T. D. Vu, and J. J. Kuffner, “Autonomous behaviors for interactive vehicle animations,” Graphical Models, vol. 68, no. 2, pp. 90–112, 2006.
[13] S. M. LaValle and J. J. Kuffner Jr, “Randomized kinodynamic planning,” The international journal of robotics research, vol. 20, no. 5, pp. 378–400, 2001.
[14] H. Liu, X. Zhang, J. Wen, R. Wang, and X. Chen, “Goal-biased bidirectional rrt based on curve-smoothing,” IFAC-PapersOnLine, vol. 52, no. 24, pp. 255–260, 2019.
[15] Z. Tahir, A. H. Qureshi, Y. Ayaz, and R. Nawaz, “Potentially guided bidirectionalized rrt* for fast optimal path planning in cluttered environments,” Robotics and Autonomous Systems, vol. 108, pp. 13–27, 2018.
[16] W. Xinyu, L. Xiaojuan, G. Yong, S. Jiadong, and W. Rui, “Bidirectional potential guided rrt* for motion planning,” IEEE Access, vol. 7, pp. 95034–95045, 2019.
[17] O. Khatib, “Real-time obstacle avoidance for manipulators and mobile robots,” in Autonomous robot vehicles. Springer, 1986, pp. 396–404.
[18] J. hwan Jeon, S. Karaman, and E. Frazzoli, “Anytime computation of time-optimal off-road vehicle maneuvers using the rrt,” in 2011 50th IEEE Conference on Decision and Control and European Control Conference. IEEE, 2011, pp. 3276–3282.
[19] D. J. Webb and J. Van Den Berg, “Kinodynamic rrt*: Asymptotically optimal motion planning for robots with linear dynamics,” in 2013 IEEE International Conference on Robotics and Automation. IEEE, 2013, pp. 5054–5061.
[20] Y. Li, Z. Littlefield, and K. E. Bekris, “Asymptotically optimal sampling-based kinodynamic planning,” The International Journal of Robotics Research, vol. 35, no. 5, pp. 528–564, 2016.
[21] A. Sivaramakrishnan, Z. Littlefield, and K. E. Bekris, “Towards learning efficient maneuver sets for kinodynamic motion planning,” arXiv preprint arXiv:1907.07876, 2019.
[22] Z. Littlefield and K. E. Bekris, “Informed asymptotically near-optimal planning for field robots with dynamics,” in Field and Service Robotics. Springer, 2018, pp. 449–463.
[23] Z. Littlefield and K. Bekris, “Efficient and asymptotically optimal kinodynamic motion planning via dominance-informed regions,” in 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). IEEE, 2018, pp. 1–9.
[24] B. Akgun and M. Stillman, “Sampling heuristics for optimal motion planning in high dimensions,” in 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems. IEEE, 2011, pp. 2640–2645.
[25] C. Urmon and R. Simmons, “Approaches for heuristically biasing rrt growth,” in Proceedings 2003 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2003)(Cat. No. 03CH37453), vol. 2. IEEE, 2003, pp. 1178–1183.
[26] I. Palunko and R. Fierro, “Adaptive control of a quadrotor with dynamic changes in the center of gravity,” IFAC Proceedings Volumes, vol. 44, no. 1, pp. 2626–2631, 2011.
[27] R. Carona, A. P. Aguia, and J. Gaspar, “Control of unicycle type robots tracking, path following and point stabilization,” 2008.
[28] L. E. Kavraki, P. Svestka, J.-C. Latombe, and M. H. Overmars, “Probabilistic roadmaps for path planning in high-dimensional configuration spaces,” IEEE transactions on Robotics and Automation, vol. 12, no. 4, pp. 566–580, 1996.
[29] M. Kalisiak and M. van de Panne, “Rrt-blossom: Rrt with a local flood-fill behavior,” in Proceedings 2006 IEEE International Conference on Robotics and Automation, 2006. IEEE, 2006, pp. 1237–1242.
[30] N. J. Sanket, C. D. Singh, C. Fermüller, and Y. Aloimonos, “Prgflyt: Ai based indoor quadrotor autonomy framework for navigation and interaction tasks,” https://github.com/prgumd/PRGFlyt/wiki, 2019.

APPENDIX

Algorithm 6: RRTSingleIteration($G, E, V$

1. $E_{new} \leftarrow$ getEdgeExtend($G$)
2. $v_{new} \leftarrow E_{new}\text{-finalNode}()$
3. if not CollisionCheck($E_{new}$) then
   4. $V \leftarrow V \cup \{v_{new}\}$
   5. $E \leftarrow E \cup \{E_{new}\}$
   6. return $v_{new}$
7. return NULL

Algorithm 7: getEdgeExtend($G$

1. $v_{next} \leftarrow$ RandomConfig()
2. $v_{near} \leftarrow$ NearestNeighbor($v_{next}, G$)
3. $v_{new} \leftarrow$ Extend($v_{near}, v_{next}$)
4. return ($v_{near}, v_{next}$)