Motion Planning Around Plants

Anirudh Aatresh
Dept. of ECE
University of Michigan
Ann Arbor, USA
aaatresh@umich.edu

Johnson Zhong
Robotics Institute
University of Michigan
Ann Arbor, USA
zhsh@umich.edu

Instructor: Dmitry Berenson
Robotics Institute
University of Michigan
Ann Arbor, USA
dmitryb@umich.edu

December 20 2021

***Abstract*—Robot gardening with a manipulator will often involve scenarios where the arm will need to move through plants for the end effector to reach its target position without damaging them. Success of rapidly exploring random trees (RRT) motivate us to develop a modified bi-directional RRT framework capable of gently deflecting plants out of the arm's way as it reaches the goal configuration from the initial state. We define a cost function metric to assess our method, and our experiments with various plant based environments show that our method yields the lowest score in each environment when compared to benchmark techniques. We also extend this effort towards planning under uncertainty and demonstrate our results in this scenario as well. Our results show that the algorithm developed during this independent study is capable of producing suitable paths through plants to reach its destination in complex plant environments.**

***Index Terms*—Motion Planning, Robot Gardening.**

I. Introduction

The task of motion planning for manipulators around plants is an especially time-worthy one as the dexterity and degrees of freedom offered by manipulators can be incredibly useful for a variety of gardening tasks. The ability to reach various locations and achieve complex configurations permits a manipulator's end effector to reach areas that would be otherwise hard to get to. A great application for such a robot would be weed removal and produce harvesting, where the object of interest would at times be at the base or in between tall plants, and would require a robot to move through these plants to get to it. This would be a requirement for the robot garden project on the roof of the Wilson Center, University of Michigan, where a mobile robot with a manipulator will be required to carry out gardening functions.

The task of moving gently through plants to get to an object of interest is challenging as the planner must somehow keep track of the state of the plants to prevent damaging them as it finds a path to its goal from its initial state. We are motivated to solve this problem as it can enable a manipulator to move through difficult environments in order to get to its final state, without damaging any of the plants in its way. The environment in this context would be a garden in which plants are present along with a particular object of interest that we would like to reach using a manipulator. In our experiments, we carry out these tasks in simulation and validate our ideas by comparing it against two benchmark approaches that we later develop.

In addition to solving this motion planning problem, we consider the task of planning under uncertainty. Simulation experiments when mimicked in the real world often fail to reliably mirror the simulation results when tested under similar conditions due to the inability of the simulator to produce an environment that is close to real-world conditions. However, such experiments with uncertainty in environments are crucial as they give us a more realistic picture about the kind of unpredictable environments the planner will see in the real world.

The motion planning problem is a well studied task in robotics. In its essence, it is being able to find a path from where the robot is now, to where it should be such that it does not *collide* with other parts of itself and the environment. Using mathematical notation, it can be defined as: Given a robot description $\mathcal{A}$, obstacle description $\mathcal{O}$, a C-space $\mathcal{C}$, initial configuration $q_{init} \in \mathcal{C}$ and a goal configuration $q_{goal} \in \mathcal{C}$, compute a continuous path $\tau : [0,1] \rightarrow \mathcal{C}_{free}$ such that $\tau(0) = q_{init}$ and $\tau(1) = q_{goal}$ [1].

One of the earliest successful attempts at deploying motion planning algorithms in the real world was on *Shakey* [2]. *Shakey* demonstrated this excellently for its time, where it used a visibility graph search algorithm to plan its route. Today, motion planning algorithms have been deployed in a variety of robotics applications spanning from manufacturing robots [3] to field robots [4].

Motion planning methods can be classified into combinatorial planning and sampling based planning methods, with the later being used in more practical scenarios currently. Sampling based planning methods have gained prominence for practical implementations as they do not require explicit constructions of the C_{obs} space, which was one of the hurdles in the use of combinatorial methods. They simply work by sampling the C-space instead. Moreover, the ability to modularize various components of the planner in the sampling case such as treating the collision checker as a black box provides a higher flexibility and ease in its development.

Examples of popular sampling based planning algorithms are probabilistic roadmaps [5] and rapidly exploring random trees [6]. The probabilistic roadmap method, which is a multi-query approach, consists of two phases: the preprocessing phase and query phase. The former phase comprises of creating a graph by incrementally sampling the C-space and attempting to reach it from the nearest node in a graph. The latter phase consists of finding the shortest path through the graph between two given nodes, q_{init}, the initial node and q_{goal}, the goal node. The RRT algorithm however, is a single query approach and is of more interest to us for solving the problem at hand.

The rapidly exploring random tree algorithm [6] is a highly successful sampling based algorithm that is widely used for motion planning when the C-space is of limited dimension. It is based on randomly sampling nodes in the C-space of the robot and attempting to make a connection to that node from the nearest node in a graph that grows as the algorithm progresses. Repeatedly running this algorithm would ideally result in the planner exploring the entire C-space in a random fashion and eventually finding a path between the goal and the source or returning a null solution to the path finding problem. RRTs however, do not comply with the notion of completeness defined for combinatorial planning methods. They abide by a weaker notion of completeness, called probabilistic completeness. This means that with enough number of samples, the probability of finding a path, provided it exists, converges to one.

The bi-directional variant [7] of this algorithm consists of two trees growing in parallel, one from the source node towards the goal and the other from the goal node to the

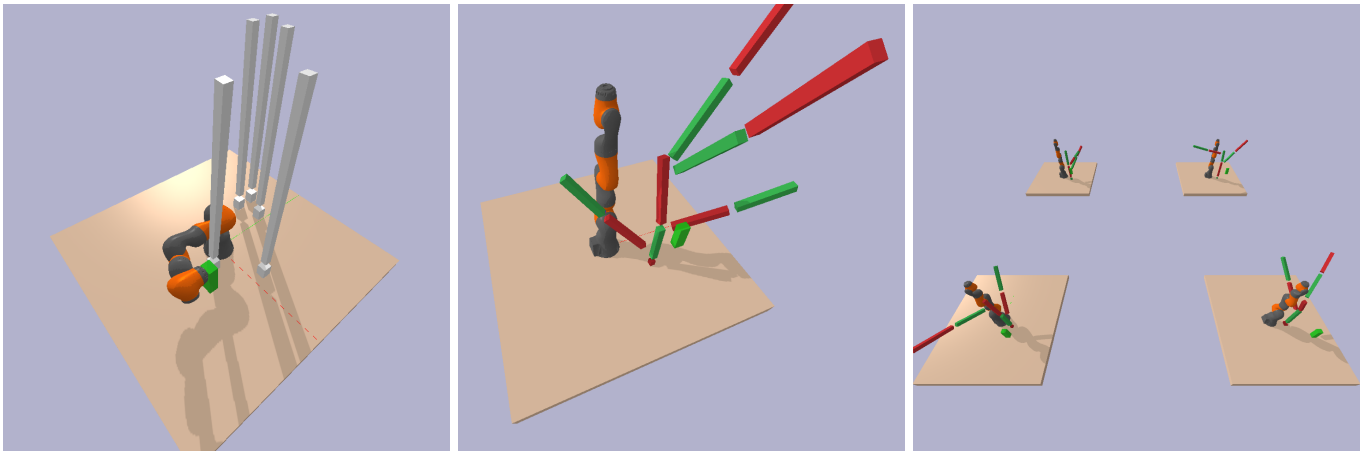

Fig. 1. Our simulation experiments. (Left) Planning around single stem plants. (Middle) Planning with a multi-branch plant. (Right) Planning in a multi world environment with multi-branch plants.

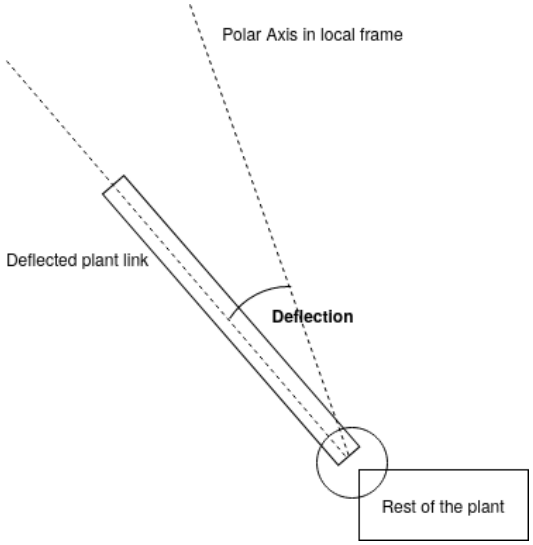

Fig. 2. Characterizing the state of a plant link using deflection angles.

source node. A combination of this bi-directional framework with the RRT connect modification [8] makes the planner robust against the infamous bug-trap and narrow passage environments. This algorithm has been described in algorithm 1. A famous accomplishment of this algorithm was its ability to solve the Alpha 1.0 puzzle in 2002. Diankov et al. in [9] proposed a bi-space planning algorithm that runs on a similar idea for concurrent exploration in differing C-spaces. We adopt a similar idea to the bi-space planning framework when developing our algorithm, with the inclusion of the storage of the states of the environment in the planning process and different criteria for constraint violations for the forward and backward trees.

Motion planning for robotic manipulators around plants has also been explored to a certain extent by the research community. Bao et al. [10] proposed the use of probabilistic roadmaps for the purpose of performing leaf phenotyping in real-time using a robotic manipulator. Their algorithm uses the voxel data generated by a Kinect sensor and laser profilometer to perform collision checking online. Ohi et al. [11] suggested using visibility graphs for the motion planning task of an autonomous pollinator robot, that is capable of performing SLAM, perception and manipulation to achieve its objective. The authors in [12] proposed a very interesting idea of using artificial potential fields [13] for the task of harvesting using manipulators, which is one of the applications that we are targeting for our algorithm. However, these references do not deal with the gentle deflection of plants by a robot to get to its final goal configuration. In our experiments, we focus on this part of the problem as it can so happen that this scenario is encountered often during robot gardening using a manipulator.

Planning in dynamic environments and under uncertainty is of prime interest among roboticists as that presents conditions which most closely resemble what planning in a real world scenario would look like. Simulators are often unable to depict physical parameters accurately, thereby leading to an improper transfer from simulation to real. This gap is referred to as the reality gap [14]. The EMPPI [15] framework is an idea proposed by Abraham et al. that can be used to produce control signals in uncertain environments. It uses an ensemble of MPPI [16] control frameworks to perform online adaptations in dynamic environments. Our approach is similar to the EMPPI framework in the sense that we use a single planner to simultaneously find a path through randomly generated environments using a custom policy.

This document has been divided into three sections, with section I covering the introduction. Section II describes the proposed algorithm, the experiments and observations that we obtained. Section III portrays the results that we have obtained and finally, section IV concludes this report.

II. Our Planning Algorithm

To tackle the motion planning problem around plants, we consider the previous success of sampling based planning algorithms such as the rapidly exploring random tree (RRT) [6] and

its modifications such as RRT connect [8] and Bi-directional RRT [7]. We further build upon them by incorporating the state of the plants into the algorithm along with path checking mechanisms to make sure the final path generated does not violate any constraint. This yields a robust planner capable of finding a path even through the most difficult of plant environments. A description of this algorithm and its working has been explained in the following subsections.

A. Characterizing the state of the plant

In order to make sure the arm does not damage any plant as it executes its path, the algorithm needs to somehow track the state of the plants in the environment. In order to keep track of its state, we must first characterize a plant by finding a measure of the amount of "damage" it incurs or the "gentleness" factor that the planner must adhere to.

In our experiments, we consider the deflection of the plant to be this measure. The deflection of a plant encompasses the deflections of all the links that make up the plant. We measure the deflection of a link as the extent of deviation from the polar axis of the link in its local frame (fig. 2). This would correspond to the polar angle measured with respect to the polar axis of the link in its local frame. We use these deflections to get an idea of the amount each plant link can be moved by the planner as it finds a path from start to goal configurations.

We define *constraint violations* as a combination of collisions of the arm with rigid objects and over-deflections of plants by the arm. Our planning algorithm must check for constraint violations as it searches and finds a path through the joint space of the arm. To measure the amount of deflection of each link, we use the dynamics of the simulator and its corresponding API in our implementation.

B. The Planner: Modified Bi-Directional RRT Algorithm

Our observations from initial experiments with the uni-directional RRT algorithm for this application showed that it is very slow and sometimes unable to converge to a solution given the time and computational resources present when this experiment was conducted. This can be attributed to the fact that the problem at hand is analogous to a narrow passage problem in its C-space, which RRT connect and bi-directional RRTs are better at solving.

In order to use this algorithm (bi-directional RRTs) for planning around plants on the condition that the robot is allowed to deflect plants within a particular limit only, one of the most significant modifications is the storage of the state of the environment in the RRT algorithm's nodes. This is important because every node in the RRT graph represents joint configurations of the robot that permit collision free and over deflection free robot states, and the state of the entire environment at that point during simulation. This is useful during the RRT planning process when an attempt is made to find a path between the closest node on the graph and the target node. Once the nearest node is found, the state of the environment stored in the nearest node is restored before attempting to extend towards the target node. The reasoning behind this operation is because the state of the environment in our experiments changes as the planner attempts to find a path, meaning that each node on the graph will be associated with a particular environment state. It is therefore crucial that this state be restored before an attempt is made to extend from it towards the newly sampled node. The new modified bi-directional RRT algorithm has been shown in algorithm 2.

In our problem formulation, we have simplified the planning task by assuming that the planner is given the initial and goal configurations to begin with, such that they do not violate any constraints. Obtaining these configurations is a trivial matter, and in our experiments, the joint state of the arm was set to the given pose using the simulator's API. The state of the

Algorithm 1: Bi-directional RRT algorithm.

Data: q_{init}, q_{goal}.
Initialization: fwdTree.init(q_{init}), bwdTree.init(q_{goal}), NUM_RRT_ITER
while $k < NUM\_RRT\_ITER$ **do**
  q_{target} = SAMPLE()
  $q_{nearest}$ = fwdTree.nearestNeighbor(q_{target})
  q_{last}, _ = EXTEND(q_{target}, $q_{nearest}$)
  $q_{nearest}$ = bwdTree.nearestNeighbor(q_{last})
  q_{bwd_last}, success = EXTEND(q_{last}, $q_{nearest}$)
  SWAP(fwdTree, bwdTree)
  IF(success):
  THEN BREAK
  ENDIF
  $k \rightarrow k + 1$
end
path = retracePath(fwdTree, bwdTree, q_{last}, q_{bwd_last})
path = smoothPath(path)
Result: path; This is the smoothed path between q_{init} and q_{goal} obtained by the bi-directional RRT algorithm.

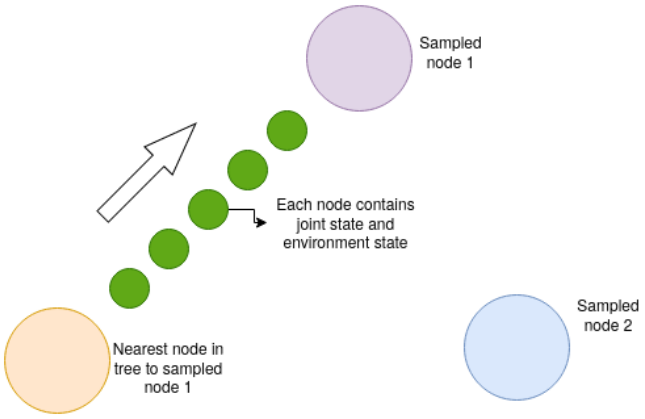

Fig. 3. Extension of the nearest node towards the newly sampled node (sampled node 1) in our modified bi-directional RRT algorithm. Samped node 2 is the next newly sampled node after an extension attempt is made toward sampled node 1.

Algorithm 2: Modified Bi-directional RRT algorithm for motion planning around movable plants for a manipulator.

Data: q_{init}, q_{goal} such that q_{init} and q_{goal} do not violate any constraints.
Initialization: q_{init}.storeEnvState(), q_{goal}.storeEnvState(), fwdTree.init(q_{init}), bwdTree.init(q_{goal}), NUM_RRT_ITER
while $k < NUM\_RRT\_ITER$ **do**
  q_{target} = SAMPLE()
  $q_{nearest}$ = fwdTree.nearestNeighbor(q_{target})
  $q_{nearest}$.restoreState()
  q_{last}, _ = EXTEND_AND_STORE_STATE(q_{target}, $q_{nearest}$)
  $q_{nearest}$ = bwdTree.nearestNeighbor(q_{last})
  $q_{nearest}$.restoreState()
  q_{bwd_last}, success = EXTEND(q_{last}, $q_{nearest}$)
  IF(success):
  THEN allFine, path = CHECK_FULL_FWD_PATH(fwdTree, bwdTree)
    IF(allFine):
    THEN BREAK
    ENDIF
  ENDIF
  $k \rightarrow k + 1$
end
path = retracePath(fwdTree, bwdTree, q_{last}, q_{bwd_last})
path = smoothPath(path)
Result: path; This is the smoothed path between q_{init} and q_{goal} such that it safely traverses without violating any constraints.

environment when the arm was at these configurations was stored in their respective nodes for future use. A forward and backward tree was created and initialized with the source and goal nodes respectively.

A node (q_{target}) is then sampled in the C-space of the manipulator and the nearest node in the forward tree is found ($q_{nearest}$). The state of the environment stored in this node is then restored. An attempt is made to extend towards q_{target} from $q_{nearest}$ by creating nodes in between them on a straight-line path at a particular resolution. The extension from $q_{nearest}$ is carried out until q_{target} is reached or a constraint is violated (fig. 3). It is very important to note that at every node, the state of the environment for that configuration is stored in that node. So to be clear, the configuration of the arm and the state of the environment at that configuration is stored in the node. The last node created in the extension process is called q_{last}. This process is done by EXTEND_AND_STORE_STATE().

In the backward tree computation however, it is not important to keep track of over-deflections of plants as the knowledge of over-deflection of a plant in the backward tree does not provide any information on how it would behave when traversed in the forward direction, which is more relevant to our problem. Hence, a simple extension operation is performed by EXTEND() which simply ignores the presence of the plants, and only checks for rigid body collisions. The purpose of the backward tree would then be to serve as a compass for the forward tree to grow towards. An important point that is brought about by the behavior of the backward tree is that these RRT graphs are direction variant, meaning that a forward traversal on an edge is not equivalent to the reverse traversal on that edge. Direction invariance on edges would be the case for simple collision checking, but for our case where more complex constraints are to be checked, this invariance does not hold. Our algorithm will therefore be complete only when we have checked for any constraint violations in the forward direction of each edge, which has been explained in more detail later below.

In this procedure, the backward tree is grown with q_{last} as its target node. The "success" of this step is monitored and decided by whether $q_{bwd\_last} = q_{target}$. If its a success, that means the forward and backward trees are connected by q_{bwd_last} and a path exists between q_{init} and q_{goal}. Now it must be noted that although a path exists, it may not be valid as the forward path along the backward tree (reverse direction along the backward tree) may not be valid. This checking is done by CHECK_FULL_FWD_PATH, which traverses the backward tree in the reverse direction (from q_{bwd_last} to q_{goal}). If any constraint violation is found, the forward tree is grown up to the node where the violation occurred, excluding it. The node that caused the violation is deleted and the remainder of the backward tree is retained. It returns the status of the operation and the entire forward path if it exists. If the status of the operation is true, that means a valid forward path exists and a smoothing operation is performed on the noisy path before returning it. If not, then the above steps are continued until success. The smoothed result would be the final output of our algorithm, representing a valid path between source and goal such that plants are deflected to get to the goal, but within their limits.

In a gardening scenario, the manipulator must be able to reach its goal configuration from its initial configuration and then find a path to its next goal or back to its initial configuration, corresponding to picking the object of interest and dropping it in a box. This algorithm can be run for those scenarios as well, by varying which configuration is the initial state and which one becomes the final state.

C. Planning under uncertainty

The planning task mentioned previously has been extended to planning under uncertainty in an effort to make it more difficult for the planner. We created the task of planning using a single planner simultaneously in multiple parallel random worlds under a particular policy.

Multiple worlds were randomly generated during run-time, thereby introducing uncertainty. The randomness that can be introduced here are regarding the placement of the branches, their natural deflections, the placement of the plant, the number of branches per stem, the number of vertical stems etc. The planner sees environments that it has never seen before and

Environment	Cost	Ignore All (Type 1)	Avoid All (Type 2)	Our method
Env1	Path Cost	**1.8121 (0.0426)**	Undefined	3.8783 (1.3698)
	Deflection Cost	4.2711 (0.8146)		**0 (0)**
	Total Cost	6.0832 (0.8572)		**3.8783 (1.3698)**
Env4	Path Cost	**1.6105 (0.0521)**	1.7586 (0.0524)	1.6975 (0.0256)
	Deflection Cost	9.7681 (35.4545)	**0 (0)**	**0 (0)**
	Total Cost	11.3786 (35.5066)	1.7586 (0.0524)	**1.6975 (0.0256)**
Env5	Path Cost	**1.5964 (0.0153)**	1.8823 (0.4802)	1.9393 (0.1386)
	Deflection Cost	2.7168 (0.6212)	**0 (0)**	**0 (0)**
	Total Cost	4.3132 (0.6365)	**1.8823 (0.4802)**	1.9393 (0.1386)

TABLE I

COMPARISON OF OUR ALGORITHM AGAINST BENCHMARKS FOR A SINGLE STEM SINGLE WORLD SCENARIO FOR $\alpha = 0.1$ AND DEFLECTION LIMIT 0.3 RAD.

Environment	Cost	Ignore All (Type 1)	Avoid All (Type 2)	Our method
Env1	Path Cost	1.7145 (0.1222)	1.8796 (0.1498)	**1.6210 (0.1142)**
	Deflection Cost	2.8535 (3.7634)	**0 (0)**	**0 (0)**
	Total Cost	4.5680 (3.8856)	1.8796 (0.1498)	**1.6210 (0.1142)**
Env2	Path Cost	**1.5831 (0.0513)**	2.7836 (1.0559)	2.2128 (0.1464)
	Deflection Cost	4.4401 (4.2118)	**0 (0)**	**0 (0)**
	Total Cost	6.0232 (4.2631)	2.7836 (1.0559)	**2.2128 (1.7271)**
Env3	Path Cost	**1.6372 (0.0513)**	Undefined	3.9327 (1.2501)
	Deflection Cost	3.6096 (8.4058)		**0 (0)**
	Total Cost	5.2468 (8.4571)		**3.9327 (1.2501)**
Env4	Path Cost	**1.6974 (0.1257)**	1.7623 (0.2023)	1.7188 (0.2052)
	Deflection Cost	2.6073 (14.8046)	**0 (0)**	**0 (0)**
	Total Cost	4.3047 (14.9303)	1.7623 (0.2023)	**1.7188 (0.2052)**

TABLE II

COMPARISON OF OUR METHOD AGAINST BENCHMARKS FOR A MULTI-BRANCH SINGLE WORLD SCENARIO WITH $\alpha = 0.1$ AND DEFLECTION LIMIT 0.3 RAD.

tries to find a path from source to goal using a single planning algorithm under a predefined policy. In our experiments, we have considered a pessimistic policy, where a violation in any one of the worlds is considered a violation in all the worlds. To be more clear, we adjust this strictness to obtain two policies that we further test and compare against benchmark approaches. The first of these policies enforces strictness 95% of the time, meaning that 5% of the constraint violations are not reported to the planner. The second policy that has been used is when the strictness is reduced to 30%, which is much more lenient than the first policy. The choice of this kind of sampling based policy can be justified by relating it to the epsilon greedy approach in reinforcement learning, where a trade-off is reached between exploration and exploitation.

D. Experimental Conditions, Comparison and Metrics

1) Software and technologies used: The simulator that we have used in all our experiments is PyBullet [17]. PyBullet provides an excellent simulation environment to test motion planning algorithms in various obstacle laden worlds. The programming and implementation was done in Python with the help of its associated libraries.

2) Creation of a plant in simulation and robot used: To get a good idea about the performance of our planning algorithms, we must first create a plant in simulation that behaves similar to how a plant would behave when deflected, or when an external force is applied on it. We used the Unified Robotic Description Format (URDF) to create an initial model of a single stemmed plant 1. Plant link joints were created with two revolute joints permitting movement in two axes. More complex plants with multiple stems, branches and natural deflections were created programmatically using the PyBullet's API. Programmatic generation gives us the ability to create randomized versions of a plant based on the number of branches per stem, number of stems and the natural deflections of the branches and stems.

An external torque was applied on the links proportional to the amount of deflection incurred. This gives the plant link a method to recover from a deflection caused by an external force. The angular deflection was measured as the relative difference of orientations of each link, thereby resulting in the relative deflection of a link with respect to its parent link. These calculated values were measured for over-deflection by comparing it against a preset deflection limit.

The robot manipulator that we have used in this experiment is the KUKA IIWA robot. The URDF model of this robot is avilable alongside the PyBullet planning library [18].

3) Benchmarks and metrics: To truly understand how well our method is performing, we compare it with two benchmark methods. The first of these two methods (Type 1), can be described as ignoring all plants to get to the goal, meaning that the planner ignores the amount of deflection each plant is undergoing. The second of these methods (Type 2) avoids all plants, that is, avoids all contact or collision with plants.

To quantify the performance of these methods, we formulate a cost function that taxes any given approach on its end effector path length and the total amount of over-deflection encountered. It can be defined as follows:

$$C(p) = \Lambda(P) + \alpha\Phi(P) \tag{1}$$

Envs	Cost	Averaged Readings Across Worlds				Worst-case Reading Across Worlds			
		Ignore All	Avoid All	Our method (95%)	Our method (30%)	Ignore All	Avoid All	Our method (95%)	Our method (30%)
Env1	Path Cost	**1.5299 (0.0018)**	Undefined	4.0304 (0.4739)	5.6194 (1.1517)	**1.5299 (0.0018)**	Undefined	4.0304 (0.4739)	5.6194 (1.1517)
	Deflection Cost	0.3212 (0.1113)		0.4260 (0.4306)	**0.0192 (0.0029)**	1.2688 (0.4453)		1.7042 (1.7227)	**0.0763 (0.0116)**
	Total Cost	**1.8511 (0.1131)**		4.4564 (0.9045)	5.6386 (1.1546)	**2.7987 (0.4471)**		5.7346 (2.1966)	5.6957 (1.1633)
Env2	Path Cost	**2.0099 (0.0338)**	3.3342 (0.2174)	2.2085 (0.0624)	2.1026 (0.1911))	**2.0099 (0.0338)**	3.3342 (0.2174)	2.2085 (0.0624)	2.1026 (0.1911)
	Deflection Cost	3.4262 (2.1600)	**0 (0)**	**0 (0)**	**0 (0)**	9.1069 (2.6022)	**0 (0)**	**0 (0)**	**0 (0)**
	Total Cost	5.4361 (2.1938)	3.3342 (0.2174)	2.2085 (0.0624)	**2.1026 (0.1911)**	11.1168 (2.6360)	3.3342 (0.2174)	2.2085 (0.0624)	**2.1026 (0.1911)**

TABLE III

QUANTITATIVE COMPARISON OF OUR MODEL WITH BENCHMARKS IN A FOUR WORLD SCENARIO WITH $\alpha = 0.1$ AND DEFLECTION LIMIT SET TO 0.3 RAD.

where, p denotes a path from the source node (q_{init}) to goal node (q_{goal}), $C(p)$ denotes the total cost of traversing path p, $\Lambda(p)$ denotes the end effector path length and $\Phi(p)$ denotes the amount of over-deflection beyond the limit. The weigting constant α is used to adjust the contribution of the over-deflection cost to the total cost.

Method	**Path length**	**Deflection over the limit**
Ignore all plants (Type 1)	Small	Large
Avoid all plants (Type 2)	Large	0
Our method	Moderate	0

TABLE IV

EXPECTED PERFORMANCE ANALYZED USING THE COST FUNCTION METRIC.

We hypothesize that their behavior must align with table IV. In this table, it can be seen that Type 1 must yield a small path length but a large over-deflection cost, thereby yielding a large total cost. Type 2 however, will yield a large path length but zero over-deflection as it avoids plants altogether, leading to a large total cost. Our method will perform the most efficient compared to these two approaches when we consider the total cost. The path length cost will lie in between type 1 and type 2, with zero over-deflection cost as no plant should be deflected beyond its limit. For environments that have either the initial configuration and/or final configuration in contact with the plant (not necessarily violating any constraint), the Type 2 benchmark cost would be undefined.

III. RESULTS

A. Planning in a single world environment

This is the planning task of moving a robotic arm from start to goal without violating any constraints in a single world. Table I shows a quantitative comparison of our method with the benchmarks for plants with single stems only, as described in figure 1. Table II describes the quantitative results that we have obtained using a multi-branch plant (fig. 1).

The results in table I have been calculated based on five trials per method per environment. The mean of the data collected in these five trials have been shown for each cost along with their respective variances in parenthesis. It can be observed that environments 1 and 4 clearly follow the pattern in hypothesized in table IV. Environment 5 shows some deviation in the mean path cost, where our method is larger than the Type 2 benchmark path cost. However, the variance of the path cost is larger for Type 2, with our method having the lowest. Therefore, this deviation in path cost can be explained by a lower number of trials used in this experiment and the probabilistic nature of the RRT. Some video recordings of our method at work in this scenario can be found here: https://youtube.com/playlist?list=PLoBFj4IxcH77DDflsiRRq8lKDZOxAwiEm.

Similarly, table II shows results for a multi-branch single world scenario with twenty trials conducted per method per environment. In other words, these results were obtained when our planner was tasked with planning around plants with multiple stems and multiple branches. The plants' shape and configuration were generated randomly. Our method obtains the lowest mean total cost when compared against the benchmarks. The variances are not the lowest for environments 2 and 4, but they are quite comparable to the lowest in each environment. Some video recordings of our simulation experiments for the multi-branch case can be found here: https://youtube.com/playlist?list=PLoBFj4IxcH76gsr22tLmSk00yhLAzZxqt.

B. Planning in a multi world environment

For the case of planning in a multi world environment, we created a simulation of multiple parallel worlds, comprising of randomly generated plants. All the parallel worlds use the same planner to find a path from start to goal, but due to the uncertainty in each world, they may encounter constraint violations at different time steps. To deal with this, we define a policy that helps us make a decision. The environments created in our experiments were easy in the level of difficulty as more complex environments require much higher memory with our current implementation. In these experiments, we have fixed the number of branches per stem and number of vertical stems. However, in the future, these should be ideally randomized as well to create truly uncertain complex environments.

As mentioned earlier, we follow a varying pessimistic policy. We have used two levels of pessimism, 30% and 95%, and we compare them against the previously defined benchmarks, in a multi world scenario. For Type 1 method, the policy would be to ignore all plants in every world. However, for Type 2, the policy would instead be to avoid all plants in all worlds. This would sometimes be impossible as there could be no way of finding a valid path in two worlds when a valid path exists only in one and no valid path at all in the other.

In the results shown in table III, four worlds have been considered with three trials per environment per method. The mean and variance for each method has been shown in the table. The averaged readings comprise of averaging the costs across the worlds and the worst-case readings comprise of the largest costs among all worlds. Environment 2 is in line with our hypothesis. The Type 1 benchmark occupies the least path

cost among all methods. The deflection costs are zero for Type 2 benchmark and our methods as well. It can be noticed that even though the strictness has been reduced, the planner does not try to make use of it in this case and attempt to reach the goal in a shorter path with over-deflections. These costs are dependent on the kind of environment the planners are tested on, and as this is an easy environment, there will be less pressure to over deflect and a zero deflection cost can happen more frequently than with a complex environment. Environment 1 however slightly deviates from our hypothesis. In both averaged readings and worst case readings, the deflection cost is higher for our method (95%). Moreover, the path cost for our method with 30% strictness is higher than that of the 95% strictness. Environment 1 is a slightly more complex environment when compared to environment 2. The number of trials made were limited to three per method per environment due to high program run-times. The results could be expected to fall in line with out hypothesis with a larger number of trials. With the presence of additional time, this could be tested for a higher number of worlds and more complex environments. Video data depicting simulation experiments for the multi world case can be found here: https://youtube.com/playlist?list=PLoBFj4IxcH76rV1UOF0yK0F99g6O3QsPF.

IV. Conclusion

The task of motion planning around plants has been studied in this independent study course and a planning framework was created based on the bi-directional RRT algorithm. We have shown through our experiments that our method is able to outperform benchmark approaches for different environments with varying difficulty, thereby proving its robustness. In addition to the deterministic approach, our experiments with planning under uncertainty also showed positive results with regards to our method's performance. However, one thing that is yet to be proven is our method's ability to work in the real world, and what modifications could be made to fix any discrepancies. Future work in this experiment could include testing our algorithm on more difficult uncertain environments in a multi world scenario and comparing various policies for this task. It would also include using more realistic plant models during simulations and testing it on physical hardware. Moreover, planning under uncertainty could be used alongside a shape completion algorithm from a perception system to help plan in the presence of occlusions. Lastly, a faster and more memory efficient implementation of this algorithm can be made, with a focus on more efficient loading and saving of states.

References

[1] S. LaValle, "Motion planning: The essentials," *Robotics & Automation Magazine, IEEE*, vol. 18, pp. 79–89, 03 2011.

[2] N. J. Nilsson, "Shakey the robot," 1984.

[3] J. Mirabel, F. Lamiraux, T. L. Ha, A. Nicolin, O. Stasse, and S. Boria, "Performing manufacturing tasks with a mobile manipulator: from motion planning to sensor based motion control," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, pp. 159–164, IEEE, 2021.

[4] E. Heiden, L. Palmieri, L. Bruns, K. O. Arras, G. S. Sukhatme, and S. Koenig, "Benchmarking sampling-based motion planning pipelines for wheeled mobile robots," 2021.

[5] L. Kavraki, P. Svestka, J.-C. Latombe, and M. 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.

[6] S. M. Lavalle, "Rapidly-exploring random trees: A new tool for path planning," tech. rep., 1998.

[7] M. Jordan and A. Perez, "Optimal bidirectional rapidly-exploring random trees," Tech. Rep. MIT-CSAIL-TR-2013-021, Computer Science and Artificial Intelligence Laboratory, Massachusetts Institute of Technology, Cambridge, MA, August 2013.

[8] J. Kuffner and S. LaValle, "Rrt-connect: An efficient approach to single-query path planning," in *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No.00CH37065)*, vol. 2, pp. 995–1001 vol.2, 2000.

[9] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," in *Proceedings of Robotics: Science and Systems (RSS '08)*, June 2008.

[10] Y. Bao, L. Tang, and D. Shah, "Robotic 3d plant perception and leaf probing with collision-free motion planning for automated indoor plant phenotyping," 01 2017.

[11] N. Ohi, K. Lassak, R. Watson, J. Strader, Y. Du, C. Yang, G. Hedrick, J. Nguyen, S. Harper, D. Reynolds, C. Kilic, J. Hikes, S. Mills, C. Castle, B. Buzzo, N. Waterland, J. Gross, Y.-L. Park, X. Li, and Y. Gu, "Design of an autonomous precision pollination robot," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pp. 7711–7718, 2018.

[12] L. Luo, H. Wen, Q. Lu, H. Huang, W. Chen, X. Zou, and C. Wang, "Collision-free path-planning for six-dof serial harvesting robot based on energy optimal and artificial potential field," *Complexity*, vol. 2018, pp. 1–12, 11 2018.

[13] O. Khatib, "Real-time obstacle avoidance for manipulators and mobile robots," in *Proceedings. 1985 IEEE International Conference on Robotics and Automation*, vol. 2, pp. 500–505, 1985.

[14] N. Jakobi, P. Husbands, and I. Harvey, ""noise and the reality gap: The use of simulation in evolutionary robotics,"," vol. 929, pp. 704–720, 01 1995.

[15] I. Abraham, A. Handa, N. Ratliff, K. Lowrey, T. D. Murphey, and D. Fox, "Model-based generalization under parameter uncertainty using path integral control," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 2864–2871, 2020.

[16] G. Williams, A. Aldrich, and E. Theodorou, "Model predictive path integral control: From theory to parallel computation," *Journal of Guidance, Control, and Dynamics*, vol. 40, pp. 1–14, 01 2017.

[17] E. Coumans and Y. Bai, "Pybullet, a python module for physics simulation for games, robotics and machine learning." http://pybullet.org, 2016–2021.

[18] C. R. Garrett, "Pybullet planning." https://pypi.org/project/pybullet-planning/, 2018.