Path Planning for Bulldozers with Curvature Constraints

This paper was downloaded from TechRxiv (<https://www.techrxiv.org>).

LICENSE

CC BY 4.0

SUBMISSION DATE / POSTED DATE

15-12-2021 / 17-12-2021

CITATION

Smith, Elliott; Jayakody, Hiranya; Whitty, Mark (2021): Path Planning for Bulldozers with Curvature Constraints. TechRxiv. Preprint. <https://doi.org/10.36227/techrxiv.17204237.v1>

DOI

[10.36227/techrxiv.17204237.v1](https://doi.org/10.36227/techrxiv.17204237.v1)

Path Planning for Bulldozers with Curvature Constraints

Elliott Smith, Hiranya Jayakody, Mark Whitty* *IEEE Member*, UNSW Sydney

Abstract

There is presently no solution to the problem of an autonomous bulldozer pushing mounds of material to desired goal locations in the presence of obstacles whilst obeying the kinematic constraints of the bulldozer. Past work has solved some aspects of this problem, but not all. This research presents the first complete, practical solution to the problem. It works by creating a fixed RRT in advance, and then during operation connecting pushing poses into this RRT using Bezier curves. The RRT algorithm leverages a novel data structure for performing nearest neighbour comparisons for Ackermann-steering vehicles; termed the Distmetree. The resulting pushing states are searched using greedy heuristic search to find a solution and the final path is smoothed with cubic Bezier curves. The mode of operation chosen for best performance also constructs bidirectional RRTs to reach difficult to access pushing poses. The final mode of the algorithm was tested in simulation and proven to be able to solve a wide variety of maps in a few minutes while obeying bulldozer kinematic constraints. The algorithm, whilst not optimal, is complete which is the more desirable property in industry, and the solutions it produces are both feasible and reasonable.

Index Terms—Bulldozer, Path Planning, RRT, Bezier Curve, Distmetree

I. INTRODUCTION

THE automation of the bulldozer would have positive benefits for several industries, including both the mining and construction industries. These benefits include increased safety, overcoming skilled labour shortages and improving productivity. Previous research on bulldozer automation has included work by Hirayama et al. [1] who investigated the dumping area operation. The challenge is to plan a series of straight drives by the bulldozer to push a pile of material over the dumping area edge. The authors showed progress in both mapping the terrain and planning a path to push it over the edge using A* search.

In this research, instead of focusing on the dumping area operation the bulldozer is tasked to push *several* mounds of material to specified goal location in the presence of obstacles whilst obeying the kinematic constraints of the bulldozer. Without such kinematic constraints, a bulldozer will exert undesirable forces and torques on the bulldozer and damage the ground surface. The research is an extension of past work by Swift [2], who made a significant number of simplifications to this practical problem that made it closely resemble the puzzle game Sokoban but in a continuous space.

Given that an algorithm to solve the problem this research poses does not appear in the literature, the aim of this work is to produce an algorithm that can solve the problem as efficiently as possible. The key contributions of this work are an algorithm to solve the problem, that is shown to be efficient enough to apply in an industry setting, and an algorithm for computing nearest neighbours of Ackermann-steering vehicles using an Octree-like data structure termed the Distmetree.

The rest of the paper is structured as follows. In the literature review (II) highlights previous attempts at this problem. The methodology (III) details an algorithm to solve the problem posed by this research. In the results and discussion section (IV, V) different modes of operation of the algorithm are examined, and their performance compared and contrasted. Finally, the conclusion (VI) determines the best mode of operation and proves it satisfies the aim of the research, as well as suggesting avenues for future work.

II. LITERATURE REVIEW

A. Pushing Planning in a Continuous Space

A reasonable amount of work has already been completed in the area of pushing objects to desired goal locations in the presence of obstacles.

Nieuweshnaisen, Frank van der Stappen and Overmars [3] propose a method of pushing a disk to a goal location that incorporates compliant motions. This means the disk is allowed to make contact with obstacles in the environment. Similar to this work they make use of an RRT to plan the motions of the disk. However unlike this work they do not incorporate any kinematic constraints and rely on the robot being in contact with the disk at all times. Furthermore the use of compliance is not applicable to this research due to the physical properties of material mounds.

De Berg and Gerrits [4] explained how to remove one of the constraints of Nieuweshnaisen et al.'s work by showing how to compute unrestricted push plans in which the pusher can occasionally let go of the object. They propose a novel method for determining where to release the object and where to re-position the robot to start pushing again. However they do not discuss planning algorithms and continue to rely on compliance, which is not applicable in a bulldozer setting.

Ben-Shahar and Rivlin [5] use a back-propagating Dijkstra's algorithm combined with the hill-climbing technique to solve the problem of pushing multiple disks to multiple goals. However their work was too computationally expensive to be applicable in an industry setting.

Lynch and Mason [6] focus on stable pushing which is somewhat applicable to the bulldozer setting where the blade

forms contact with the material mound. Their work presents an algorithm for planning a stable push of a single object, using best-first search on a variation of Dijkstra's algorithm applied to the free configuration space. However the limitation of the algorithm is that it only works for a single object and only plans a path to a goal neighbourhood as opposed to a precise location.

Some of the most ambitious and practical work in the field was completed by Meriçli, Veloso and Akin [7]. They propose a method for pushing complex objects such as chairs and shopping trolleys by developing experimental motion models. Of relevance to this research is their use of a planning algorithm that incorporates RRTs and the similarity metric they use which is similar to the Distmetree this research presents. Furthermore, their algorithm was tested in a practical environment, and whilst re-planning was required, the results were promising. However, they still make the simplification of an omni-directional robot, a constraint that this research eliminates.

B. RRTs

An RRT is the underlying low-level path planning data structure used in this work. Whilst a basic implementation of the algorithm was used in this research[8], this section will also examine recent advancements that may be worth investigating to improve the time complexity or path length as part of future work and highlight limitations of past research.

LaValle and Kuffner present the first RRT algorithm [8], which was used as the template for the RRT algorithm in this work. Their work is not formulated to handle both ordinary and pushing motions however one important contribution of the paper is its choice of vehicle model and use to derive a nearest neighbour function. Here, we leverage this idea to define an Ackermann-steering vehicle model and distance metric that is used to find nearest neighbours by the Distmetree algorithm.

Salzman and Halperin [9] present LBT-RRT which blends the RRT and RRT* methods using an approximation factor. Their work contains useful pseudo-code and implementation details for the method of this research as well as detailing how to apply A* search to the resultant trees. Zhu *et al.* [10] demonstrate an algorithm for building RRTs at run time, which was proven to be fast and reliable and may therefore have relevance for any practical experiments conducted in the future.

Finally, Shi *et al.* [11] show how to plan and track the trajectory of an autonomous wheel loader, a vehicle that is often used in a similar context to a bulldozer. They use RRT* to plan a trajectory and the CC Steer algorithm to smooth the path according to the minimum turning radius of the wheel loader. This is similar to how this research uses Bezier curves to smooth RRT paths whilst obeying a minimum turning radius of the bulldozer. However there are significant differences between a wheel loader and a bulldozer, and therefore further work was required to incorporate pushing with these ideas.

C. Bezier Curves

Past work by Choi, Curry and Elkaim [12] explains the fundamentals of Bezier curves and how they can be applied

to an autonomous vehicle setting. Importantly they explain how to compute curvature. This work uses these equations and the fact that radius of curvature is the inverse of curvature, to compute Bezier curves for the bulldozer that obey the nonholonomic minimum radius constraints placed on the bulldozer as explained in Section III-A.

Further work by Marcharet, Neto and Campos [13] contains the equations used to generate the cubic Bezier curves used to smooth the RRT paths in this work. Finally work by Cimurs, Hwang and Suh [14] explains how to smooth paths from another path planning routine (e.g. RRT) using cubic Bezier curves and was used as part of the method of this research. An important contribution it makes to the method is the suggestion of using the golden section search method for locating the point of minimum radius of curvature of a cubic Bezier curve. This technique was adapted slightly for use in this research.

D. Bulldozer Path Planning

Swift [15] aimed to solve a similar problem to this research; how autonomous bulldozers can push mounds of material to desired goal locations in the presence of obstacles. However he makes a number of simplifications to the problem that make it closely resemble the puzzle game Sokoban, albeit transformed to a continuous space. The bulldozer is considered to be a disk shaped robot that can only move forward or turn on the spot. The mounds of material and goals are both disks and the disks and robot are allowed to move in and out of the goals. Finally his solution involves only straight line vehicle motions.

In conclusion, none of the past approaches shown throughout this literature review present a complete solution to the problem of push planning in a continuous space whilst avoiding obstacles and obeying the kinematic constraints of the bulldozer.

The aim of this research is therefore to build upon Swift's work by removing some of the unrealistic assumptions. The disks are not allowed to be pushed out of the goals as this would not be possible using a bulldozer moving dirt in practice. In addition, this work adopts a more practical vehicle model of a bulldozer, by limiting the radius of the turns it can make, such that it can no longer turn on the spot. Finally, this research replaces Swift's visibility graph with an RRT, allowing it to operate in a fully continuous space.

III. METHODOLOGY

The methodology is broken down into the following sections. In III-A the vehicle model that represents the change in approach taken by this research is explained. In III-B we explain the data structures and algorithms used to enable exploration of the map and connection into points where pushing motions can be applied. In III-C the collision avoidance algorithm is detailed. In III-D we detail how a greedy search can be applied over the possible pushing motions to push the disks into the goals. In III-E we explain how to connect the pushing poses into the RRT using Bezier curves. In III-F we outline the two heuristics tested for the greedy search. Finally, in III-G we explain how to use a low-level A* search to navigate the bulldozer through the RRT to the push points;

and in III-H how to smooth the final solution paths with cubic Bezier curves.

The operational workflow of the proposed methodology is presented in Fig. 1. The following assumptions were relied upon during the development of the method. They were chosen either to make the problem tractable (shapes of objects in the environment); to represent the real world scenario (number of disks/goals, inability to push out of goals and kinematics constraint); or to reflect past work in the field (mechanics of pushing):

- The material mounds, goals and bulldozer are disks with the same radius
- There is an equal number of material disks and goals
- Each goal can only contain a single material disk
- No material disk can be pushed out of its goal (the goals are assumed to be holes which are filled with the mounds of material)
- The obstacles and boundary of the environment are reduced to polygons
- Like past work in the field [3], [4], [5], [6], disks move only when actively contacted and pushed by the bulldozer, in a straight line path through their centre
- The kinematic constraint placed on the bulldozer is to limit the radius of curvature of the turns it makes, such that it cannot turn on the spot.

The sub-modules of the proposed algorithm are discussed in detail from III-A onwards and code is available online¹.

A. Vehicle Model

This paper proposes a realistic vehicle model which imposes a minimum steering angle on the motion, rather than assuming a bulldozer can turn on the spot which is not practical.

Inspired by Reeds-Shepp curves [16], a vehicle model based on minimal radius turns was implemented. First the maximal turns were modelled and then additional turning curves added in. The final heading of the vehicle after each control action was then calculated using simple geometry. The RRT algorithm combines one of these pre-determined steering actions with a randomly generated path length to determine a new node whilst also checking for collisions with static obstacles.

B. RRT-based Map Exploration

The first step of the algorithm is to build a fixed RRT with a number of nodes based on the size of the environment. Inspiration was drawn from the work of LaValle and Kuffner [8] with several novel modules introduced on top to ensure efficient exploration and path planning.

1) *Distance Metric Tree (Distmetree) Implementation:* One of the major bottlenecks of the RRT algorithm [17] is the time consumed in determining the nearest neighbour of a randomly sampled new node. The Distmetree solves this problem by storing the nearest neighbours according to the pre-defined distance metric in a tree data structure that allows for an average lookup time of $\log(n)$. It follows a similar structure

to a Euclidean-space Quadtree except that each Quadrant contains two states that each encompass half of the angular range. It is therefore an Octree. Each child halves this angular range but the number of children ensure that overall the entire 360° range remains covered.

Figure 2: A representation of the DistMetree and how it subdivides the environment

To compute the nearest neighbour of a query state we first check if the called Distmetree node is a leaf node. If it is a leaf node, it searches through its vehicle states and returns the one that is closest to the query state by the Distance Metric. Otherwise, it calls the function recursively on the closest child node (again by Distance Metric) that is not empty. By replacing the original naïve nearest neighbour function that searches through all the nodes in the tree with this function a significant speed-up in RRT build times is achieved, as highlighted in section IV-B. To the best of the authors' knowledge, this formulation of an Octree that uses the defined distance metric for comparisons is a novel contribution to the field.

2) *Quadtree for Radial Nearest Neighbours:* A simple Quadtree data structure was implemented to allow for the calculation of radial nearest neighbours when connecting Bezier curves into a chosen push point. This was based off the work of Behley et al [18] who implemented a similar function to operate on an Octree. The function retrieves nearest neighbours from the Quadtree within a given radius of a query state. The function recurses down through the tree collecting all the radial nearest neighbours. At leaf nodes, it simply checks each of the stored vehicle states for ones within the given distance (radius) of the query state. For nodes with children, it checks each child node to see if it overlaps with the search disk. For those that do overlap, the function is called recursively on the child.

C. Collision Avoidance

When the RRT-Distmetree algorithm samples random node from the environment and considers connecting it into the nearest neighbour of the existing tree, it must ensure the connecting edge is collision-free.

There are two cases to consider when it comes to collision detection with static obstacles and map boundary. The first case involves straight line edges which represent forward and reverse movements without any steering. Here the well-established mathematical procedure of finding the

¹See <https://github.com/Smart-Robotic-Viticulture/bulldozer-path-planning>

Figure 1: The proposed bulldozer path planning algorithm.

closest points on two line segments can be used to determine collision[19]. For the second case, when the edge is the arc of a circle (non-zero steering), the shortest distance between a circle and a line can be utilised to determine collision.

In order to avoid collisions between RRT edges and material mounds, it suffices to check the two end points (poses) of the edge. Since Bezier curve [20] edges can be significantly longer in length, it is necessary to check their entire length for collisions with material mounds. This can be done by iterating over the parameterised curve and evaluating the distance between the individual points and the centre of the material mounds.

D. Greedy Pushing Search

The goal of the high level greedy search is to plan a series of pushing motions to push all disks into a goal. There is no requirement for disks to be pushed to specific goals, but once a goal is filled with a disk this action cannot be undone.

The search process can be divided into three steps. The first is the creation of pushing poses around the disks with the aid of a pushing heatmap - an offline cache of the available approach angles to each disk in different locations (III-D1). The second is the use of Bezier curves to connect these pushing poses into the RRT, such that they can be navigated to by the bulldozer (III-D2). Finally is the calculation of the resulting state after the push is performed, which creates a new state to be added to the search space (III-D3).

The search works by popping off the next state and attempting a low level path plan between its current and previous vehicle pose. If this fails then the algorithm attempts to construct a bidirectional RRT to access poses blocked by disks (III-G1). If this fails, the search continues with the next state. If the current state is the finish state it moves on to Bezier smoothing the final solution path (III-H). Otherwise, if this state has not been visited before, it is added to a cache of visited states. Then a series of resulting states are generated using the three step process detailed before (III-D1; III-D2; III-D3), and if these have not been visited, they are added to the priority queue.

1) Step 1: Selection of Push Points using Pushing Heatmap: First, if there is a current disk being pushed, the algorithm considers pushing this disk further in the same direction. Otherwise it considers pushing either the current or other disks in various directions defined by the push points created around their circumference. In order to create these push points, the algorithm makes use of a pushing heatmap that is a cache of the unobstructed approach angles into a certain point. The heatmap is shown in figure 3.

The heatmap is created in an offline fashion and is used to enumerate push points around the circumference of the disk. To ensure they are feasible, it also checks for collisions with other disks. Finally, it adds the angle that will push the disk directly towards its closest goal. This will be termed the *continuous push angle* (III-D4).

Figure 3: The pushing heatmap of map 6, which contains three static obstacles pictured in dark blue. The bright (yellow) regions are where disks can be pushed from any approach angle whereas the dark regions show where limited or no pushing angles exist.

2) *Step 2: Connect push points to RRT with Bezier curves:* Each push point is then tested to see if it can be connected into the RRT using a Bezier curve. If it is, a new state is created that is the result of pushing the disk forward from this point (see step 3).

The connection of push points into the existing nodes of the RRT enables a path to the push point to be found by the low level path planner at a later stage (III-G). This stage of the algorithm works by computing radial nearest neighbours (III-B2) or using a cache and then connecting a Bezier curve between one of the nearest neighbours and one of three starting push points. This connection is attempted using a two degree Bezier curve with the intersection method (III-E). Once a successful connection is found, we check the motion does not collide with other obstacles or disks, before adding as a new edge in the RRT.

3) *Step 3: Create resulting state:* To begin we find the closest goal to the disk being pushed. In addition we prevent the disk being pushed out of a goal if it is currently in one. The closest goal is determined using the *GoalDistanceMetric*. This metric calculates a distance from the position of the disk (x_1, y_1) and goal (x_2, y_2) , and the angle between the push point heading and the vector from the disk to the goal (θ) . This metric is defined as

$$\begin{aligned} & \text{GoalDistanceMetric}(x_1, y_1, \theta, x_2, y_2) \\ &= \begin{cases} \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} & \sin(\theta) < 0.25 \\ \sin(\theta) \times ((x_2 - x_1)^2 + (y_2 - y_1)^2) & \sin(\theta) \geq 0.25 \end{cases} \end{aligned}$$

Having found the closest goal, we then push the disk as close to it as possible by randomising a number of pushing distances and choosing the result that gets closest to the goal. Assuming the push is successful, a new edge is created in the RRT that describes the pushing motion. This will be required by the low-level path planner later (III-G). Then a new state is created, its attributes updated and added to the search space. This is how most child states in the pushing greedy search

priority queue are generated.

4) *Final Push:* If ever the disk being considered to be pushed is within 2.5 times the radius of the disks to the centre point of the closest goal, then the algorithm attempts to push it into this goal. It determines the *continuous push angle* from the angle of the vector between the disk and this closest goal. It then creates a push point on the circumference of the disk with the heading set to this angle. Next, it checks if a Bezier curve can be connected from the RRT into this push point (III-D2). After this, it proceeds in the same way as before, by adding a new edge to the RRT and creating a new state in the search space.

E. Bezier Connection into Push Points Details

1) *Bezier Curve Intersection Method:* A degree two Bezier curve was chosen for efficiency, with the start and end points of the vehicle motion used and the intersection points of the vehicle headings defining the control point. Once the Bezier curve is created, it is necessary to check that it does not, at any point, have a smaller radius of curvature than the minimum allowed. In a degree two Bezier curve the maximum curvature occurs at the point on the curve closest to the control point. The golden section search method was used to find this point and the radius of curvature evaluated there. Assuming the curve is found to have an acceptable minimum radius of curvature, it is returned to be used for connecting motions in the RRT.

F. Heuristic

Two heuristics for the greedy search were used in this research. The first is the sum of the Euclidean distances between each disk not currently in a goal and their closest unoccupied goals. The second heuristic is identical to the first, except it adds the square root of the Euclidean distance between a disk and its closest unoccupied goal to a total sum.

G. Low Level Path Planning

The low level path planning is implemented as an A* search through the RRT. The function returns true if a path between the current vehicle pose and the previous vehicle pose can be found. Otherwise it returns false. The function is called twice, with a failure the first time resulting in a bidirectional RRT being grown to access tight areas (III-G1).

1) *Bidirectional RRT Connection:* If the low level path planner initially fails, likely because all potential paths intersect with disks, then the algorithm grows a bidirectional RRT to access this tight spot. The bidirectional RRT is grown between the push point and the previous pose. In addition the algorithm takes into account the locations of all the disks. This is the main difference between this part of the algorithm and the original RRT growth, which does not avoid collisions with disks, only obstacles. The process works by maintaining two trees simultaneously, one grown forward from the starting state, and one backwards from the goal state. In addition, two Distmetrees are used for nearest neighbour comparison.

Running for a fixed number of iterations, the algorithm takes it in turn to grow either the forward or backwards tree,

alternating between using a randomly sampled state or the goal state of the respective tree. Next, a new pose is created in the same way as before, except that we also check for collisions between potential edges and disks. Then we attempt to make a Bezier connection between this new pose and all the states of the other tree. It uses the same Bezier connection method as before (III-E) and only attempts connection of states that are ‘ahead’ of this new pose. If any connection attempt is successful then the forward and backwards trees are combined and added to the global RRT, and we continue with enumerating pushing poses.

H. Bezier Smoothing of Solution Path

The algorithm smooths the individual paths of the final vehicle path if they contain more than two poses. For each individual path that meets this criteria, it first calculates the location of the disks at that point in time. Next, since the final edge is the pushing motion (which also changes the disk positions), the new path is created by smoothing the path up until the final pose and then concatenating this with the final pose. This procedure continues until the entire vehicle path has been smoothed, and the old solution path is replaced with this new solution.

Paths are smoothed by smoothing smaller and smaller subsections of the overall path. When a successful smoothing occurs, the routine goes back to trying to smooth the rest of the path, from the successfully smoothed endpoint on-wards. It terminates once it has attempted smoothing of the whole path. The results of smoothing the solution generated for map 6 are shown in Figure 4

Figure 4: The result of Bezier smoothing map 6. The scenario is pictured top left, and the result of the RRT growth in light gray top right. Bottom left shows the solution path *before* Bezier smoothing. Red paths are traditional RRT controls, any initial Bezier curves are in orange. Bottom right shows the result *after* smoothing, this time there are many more orange Bezier paths, and the overall path length is decreased.

1) *Cubic Bezier Curve Generation:* We use cubic Bezier curves when smoothing the solution paths according to the

Figure 5: The six test maps used.

method proposed by Marachet et al. [13] by varying the values of k_1 and k_2 . This process works by fixing the start and end points of the curve to be the start and end positions $(x_i, y_i), (x_g, y_g)$, and then creating two additional control points such that the curve conforms to the required starting and ending heading. These two points are defined to be

$$\mathbf{p}_1 = [x_i + r \times \cos \theta_i, y_i + r \times \sin \theta_i]$$

$$\mathbf{p}_2 = [x_g - r \times \cos \theta_g, y_g - r \times \sin \theta_g]$$

Where r is initially 10% of the Euclidean distance between the two poses, increasing by 10% until it is 100%.

Once the control points have been created, a Bezier curve is created of degree three for smoother paths. It is checked for minimum radius of curvature and collisions before being returned for use in smoothing.

IV. RESULTS

A. Original Test Maps

Pictured in Figures 5a - 5f are the six main maps used for testing. In all maps, green disks are the goals, blue disks are the material mounds, any obstacles are pictured as black polygons and the bulldozer and its heading are given in red.

B. Distmetree Results

In order to evaluate the performance of the Distmetree algorithm, the results of building the RRT using the Distmetree were compared with the naïve algorithm. The time to build the RRT was evaluated using the 6 original test maps. It is important to note that some of these maps contain obstacles, which will be relevant when discussing the results. For now

the most important feature of these maps is their size (height and width) and thus the number of unique edges constructed in the RRT. These and all other experiments were carried out on a computer with an Intel Core i5-6200U CPU, 8GB of RAM and no GPU. A table of build times for the naïve nearest neighbour search and Distmetree algorithm is shown in table I. A graph of the average time to build the RRT using the naïve nearest neighbour search method and the Distmetree method can be seen in Figure 6.

Table I: Naïve and Distmetree method RRT build time results

Num Unique Edges	Naïve		Distmetree	
	Mean (s)	Std dev (s)	Mean (s)	Std dev (s)
3200	245	1.08	178	9.42
4000	334	4.22	232	11.2
6000	587	3.98	381	38.4
7000	813	15.1	524	50.0
7200	764	10.5	463	27.1
9600	1640	20.8	1020	24.6

Figure 6: RRT build times

C. Greedy Pushing Search: Without Bidirectional RRT

This section will evaluate the performance of the greedy pushing search algorithm when run without the option of constructing bidirectional RRTs to access difficult to reach pushing poses. For these results the algorithm was run without the addition of the construction of a bidirectional RRT when the low level path planner fails (III-G1) in its usual greedy mode of operation whereby $f(n) = h(n)$. Both of the suggested heuristics (III-F) were tested and the results are summarised in the following sections. Each map was allocated a maximum time of 1 hour (3600 seconds) to be solved.

1) *Flat Heuristic*: The timing data results and the path length results of the flat heuristic are shown in table II. The timing data results are consistent for the easier maps (1-4), but show significant variation between trials for map 5 (as shown by the large standard deviation), and map 6 was unable to be solved. In terms of path length, the first two maps have consistent path lengths across all trials. Maps 3-4 show small variation in path length. Map 5 shows significant variation

in path length that is correlated with the timing data results. Again, map 6 was unable to be solved in the time permitted.

Table II: Flat Heuristic without bidirectional RRTs Results

Map Number	Timing Data [s]		Path Length	
	Mean	Std dev	Mean	Std dev
1	56.6	1.23	3.10	0
2	16.5	0.445	2.35	0
3	140	8.63	20.3	0.883
4	58.8	4.87	9.45	2.16
5	336	74.7	45.9	12.6
6	Timeout	Timeout	NA	NA

2) *Root Heuristic*: Next the heuristic used was changed to be the root heuristic. A table of the timing data and path lengths is presented in table III. The timing data results are similar to section IV-C1 except that the harder maps (3-5) run faster. Again map 6 was unable to be solved. Examining the path length results, Maps 1-2 have the same result as before. These maps are so trivial that the same solution is generated each time, and thus the same path length. Maps 3 and 5 show improvement from section IV-C1, whilst map 4 has longer average path length. However given that the run time is faster, the mode of operation is considered to be better than section IV-C1.

Table III: Root Heuristic without bidirectional RRTs Results

Map Number	Timing Data [s]		Path Length	
	Mean	Std dev	Mean	Std dev
1	57.3	2.54	3.10	0
2	17.7	1.32	2.35	0
3	134.05	5.82	18.8	4.12
4	56.2	4.39	12.9	1.79
5	295	22.5	32.3	5.89
6	Timeout	Timeout	NA	NA

D. Greedy Pushing Search: With Bidirectional RRTs

Next the possibility of constructing a bidirectional RRT to reach difficult to access areas of the map was added into the algorithm and tested with both the original flat heuristic and the root heuristic.

1) *Flat Heuristic*: Table IV shows the run time and path length results for the flat heuristic with the option of constructing a bidirectional RRT. The timing data results show that this mode of operation is consistently slower than sections IV-C1 and IV-C2 whilst still being unable to solve map 6. Similarly the path lengths are consistently longer than sections IV-C1 and IV-C2 and map 6 was unable to be solved.

Table IV: Flat Heuristic with Bidirectional RRTs Results

Map Number	Timing Data [s]		Path Length	
	Mean	Std dev	Mean	Std dev
1	57.6	1.07	3.10	0
2	17.1	0.687	2.35	0
3	153	14.9	18.4	1.53
4	79.2	20.1	14.3	2.45
5	364	14.5	49.7	10.4
6	Timeout	Timeout	NA	NA

2) *Root Heuristic*: Finally the results of the combination of the root heuristic with the option of construction of a bidirectional RRT to access difficult areas are presented in table V. The timing data shows that this mode of operation is faster than the root heuristic without the bidirectional RRT for the easy maps (1-2) but slower for the harder maps (3-6). However this is the first and only mode of operation that can solve map 6. Since map 6 represents a common scenario of multiple material mounds in the presence of obstacles, being able to solve it is considered a priority and thus this mode of operation the best.

Looking at the path length results, Map 5 presents a significant outlier, this will be discussed further in section V-B2. The other maps have similar path length to the other modes.

Table V: Root Heuristic with Bidirectional RRTs Results

Map Number	Timing Data [s]		Path Length	
	Mean	Std dev	Mean	Std dev
1	56.1	1.64	3.10	0
2	17.1	0.55	2.35	0
3	156	8.93	21.5	2.02
4	68.0	12.3	9.39	0.738
5	382	22.8	65.4	3.87
6	1070	23.1	65.3	4.76

E. Optimal Search

It was desired to test the algorithm operating under typical A* conditions whereby $f(n) = g(n) + h(n)$. To do this, the low level path plans were computed before adding states to the high level priority queue, and the resulting path lengths used as the g cost. Due to the difference in scale between pushing distances and low level paths, this was multiplied by the weight 0.1 such that the resulting equation was $f(n) = 0.1g(n) + h(n)$.

Once again the algorithm was tested against the six original test maps, and the results are presented in Table VI. The run times are significantly longer than the chosen greedy mode of operation, and more importantly only three of the maps could be solved. Examining the path lengths, with the exception of maps 1-2, which remain so trivial as to produce the same identical solution each time, the only other map that could be solved (map 3) shows no improvement in path length compared with the greedy search.

Table VI: Optimal Search Results

Map Number	Timing Data [s]		Path Length	
	Mean	Std dev	Mean	Std dev
1	109	19.7	3.10	0
2	24.2	6.38	2.35	0
3	629	13.3	21.4	3.12
4	Timeout	Timeout	NA	NA
5	Timeout	Timeout	NA	NA
6	Timeout	Timeout	NA	NA

F. Time Complexity

First, the analytical asymptotic worst case time complexity of the algorithm was determined to be

$$O((TP - \Delta) \times (E + D^2 E \log(\frac{1}{\epsilon})) + (\Delta) \times (E + ED \log_8 E + D^2 E \log(\frac{1}{\epsilon}))) \quad (1)$$

where TP = the total minimum number of pushes required to achieve a solution. This assumes only priority queue states that further the solution are expanded, i.e. that the heuristic is very good. In reality there will be some multiplying factor e.g. nTP . Δ = number of push poses that require a bidirectional RRT to be constructed to be accessed. Typically $\Delta \ll TP$. E = number of unique edges in the RRT. This is calculated as $H \times W \times 200$ where H and W are the height and width of the map respectively, in map units. D = the number of disks (equally the number of goals as this work operates under the assumption that there is an equal number of disks and goals). ϵ = accuracy of the golden section search method. Currently the code is using a tolerance of 1×10^{-5} .

It was desired to prove one aspect of Equation 1 by showing the time complexity of the algorithm with an increasing number of disks/goals, whilst keeping all other parameters (most notably the size of the map E and the distance from each disk to its closest goal TP constant). A separate set of test maps was created in which there is a line of disks behind a line of goals, with an increasing number of disks and goals in each successive map. Each disk needs a single continuous push to push it into its respective goal. These scenarios are useful twofold. Firstly, as already stated, they can be used to analyse the time complexity of the algorithm. Secondly, they simulate a common real world dumping area situation, where there is a line of material to be pushed over the dumping ground edge [1].

A graph of the results is shown in Figure 7.

The results presented can be used to prove the analytical asymptotic worst case time complexity presented in equation 1. Simplifying this equation assuming $\Delta = TP$ yields

$$O(TP \times (E + E \log_8 ED + D^2 E \log(\frac{1}{\epsilon})))$$

$$O(TP \cdot E + TP \cdot E \log_8 E \cdot D + D^2 \cdot E \log(\frac{1}{\epsilon}))$$

For this series of experiments since map size and distance from each disk to its closest goal are constant, all variables in

Figure 7: Time complexity for the algorithm with increasing numbers of disks and goals

the above equation except D are constant (the golden section search uses a constant value for ϵ). Thus it simplifies to

$$O(mD^2 + nD + p)$$

Fitting a quadratic curve to Figure 7 gives the equation

$$f(D) = 1.938D^2 + 15.12D - 4.781$$

which conforms with the analytical results.

V. DISCUSSION

A. Distmetree Algorithm

By dividing the two equations in Figure 6 for the RRT build times, the speed-up by using the Distmetree nearest neighbour search method can be determined.

$$\frac{101.8e^{0.0002893}e^x}{71.1e^{0.0000277}e^x} = 1.4318$$

Thus the Distmetree method is 1.43 times faster than the naïve nearest neighbour search. Also note that for both methods the map with 7000 unique edges took longer than the one with 7200, going against the general trend. This was because the map with 7000 unique edges (map 5) also contains an obstacle. Recall from the method that the RRT algorithm loops until it finds a non-colliding state. Therefore the true RRT build time function is a function of both the number and size of the obstacles as well as the number of unique edges to be inserted.

B. Greedy Search

1) *Timing Data:* Across the maps that are widely solvable, the greedy search algorithm with the root heuristic but without the bidirectional RRT produces the fastest run times. In particular, maps 3-5 are faster than with the addition of a bidirectional RRT. This is unsurprising as the addition of the bidirectional RRT slows down the algorithm any time a tree is grown. However, the bidirectional RRT is proven to be necessary to solve the hardest map (6), something the three other test modes could not achieve. Therefore the addition of the bidirectional RRT into the algorithm is a necessary addition if the complexity of the maps to be solved is of a higher level. Since the original goal of this research was to be able to solve scenarios that involve multiple disks/goals and obstacles, the ability to solve this map is considered to be a priority, and

therefore the suggested mode of operation for the algorithm is the use of the root heuristic with the use of bidirectional RRTs.

2) *Path Lengths:* The path length data shows that there is in fact no correlation between path length and timing data. The time the algorithm takes to run is correlated to the number of pushes the bulldozer has to make, however it is easy to imagine a scenario where only a single push is required but the low level path to reach the required push point has a considerable length.

Secondly, the path length results are not as consistent across the modes of operation. For example, for the simplest maps 1 and 2, the path length generated was the same not only across the different modes but also individual trials. These maps are so trivial that the exact same solution is produced each time, despite the use of an RRT. For the next two maps, 3 and 4, where there are multiple disks/goals but no obstacles, the path length experiences fairly small variations. Given the conclusions from the timing data, that the root heuristic with bidirectional RRT is the most versatile mode of the algorithm, the path lengths produced by this mode compared to the other modes are within an allowable range of variation. The big difference in path length occurs in map 5, where there is a single obstacle directly blocking the path to goal. What is observed in these results is a large variation in path lengths both across modes of operation, and individual trials. Further examination showed that sometimes the bulldozer arbitrarily pushes the material mound to various random positions before manipulating it towards and into the goal.

The most likely reason for this is that greedy search will try to move the material mound as close to the goal as possible, and therefore is not very good at pushing around a blocking obstacle, where the distance to goal (for either heuristic) must first increase before it can decrease. This problem is exacerbated by the pushing heatmap, which creates a more varied range of pushing angles in the vicinity of obstacles compared to a simple fixed number of push points being used each time. Thus, the ability to cache visited states effectively is lost, and states that are close but not the same are frequently revisited. The ability to generalise the nature of visited states to prevent revisiting similar states may help to fix this problem, and could be investigated as part of future work.

C. Optimal Search

The optimal search required longer times whilst still not producing shorter path lengths. Given that the purpose of the optimal search was to produce an optimal solution with the guaranteed shortest path length, the fact that greedy can do this in a shorter time proves that this optimal mode is inefficient and unnecessary. The reason for this is because most of the path length comes from the final Bezier smoothing of solution paths, which is not affected by changing the search to a proper A* search. Thus the greedy search with the root heuristic and the option of a bidirectional RRT remains the suggested mode of operation.

D. Time Complexity

It is important to remember that equation 1 presents the worst case time complexity using upper bounds for loop iterations, and as such typical performance will be faster. This also helps to explain the significant variation in results seen between individual trials. Other reasons for this variation include the conditional nature of the bidirectional RRT growth, which when not required results in faster run times. Furthermore, the root heuristic used does not order the disk pushes in these scenarios. Therefore the D disks can be pushed into their goals in any order, significantly affecting the run time and overall nature of the solution. Finally, the underlying random nature of the RRT means no two runs are ever the same, again leading to significant differences in results.

E. Comparison with Past Work

At the present time it is difficult to make comparisons to past work as the specific combination of parameters attempted in this research have not been solved before. It is possible to find past work in the area of push planning [2] or in the combined use of RRTs and Bezier curves for non-holonomic kinodynamic planning [21] but none that combine these ideas. Swift *et al.* [2] achieved run times of on average 106 seconds for the medium complexity maps they were able to solve. Comparing this with the results of this research, for the easier maps (1-4) but with the added complexity of kinematic constraints, a faster average of 74.38 seconds was achieved, with the good benchmark of map 3 needing 156.36 seconds to be solved which is only slightly slower than Swift's work.

Examining the work of Yoon *et al.* [21] who propose the SS-RRT* method for computing path plans for car-like vehicles that avoid obstacles and obey a minimum radius of curvature constraint, when expanding 7000 nodes of the RRT (a similar number to what is used for map 3) their algorithm runs in 400 seconds. This is significantly better than the 679.97 seconds achieved in this work. However their solution does not involve pushing. For an additional 41% invested time, this research is able to produce a solution that avoids obstacles, obeys a minimum radius curvature constraint and pushes mounds of material to desired goal locations.

VI. CONCLUSION

In conclusion the aim of this research to produce a solution to the problem of pushing material mounds to goal locations while obeying kinematic constraints using autonomous bulldozers has been achieved. In terms of efficiency the algorithm developed is able to solve typical maps in a few minutes, which is comparable to past work in similar areas. The path plans produced, whilst not optimal, are reasonable as a result of using Bezier curves to smooth the solution paths. The attempt to incorporate optimality into the solution was shown to fail for many of the test maps, whilst also not producing shorter final path lengths. It was therefore rejected. It is important to realise that in industry, completeness is a more desirable property than optimality, and the chosen mode of algorithmic operation was shown to be able to solve all 6 original test maps and 10 time complexity maps.

The time complexity of the algorithm was shown to be dependent on a number of parameters, including the number of priority queue states expanded, frequency of bidirectional RRT growth being required, number of unique edges in the RRT (which in turn is derived from the size of the map) and the number of disks or goals. It therefore follows that larger maps, with more pushes required to achieve a solution and several disks/goals which are hard to access (e.g. map 6) take significantly longer. However, as already mentioned, the algorithm is able to solve this map which was the main aim of this research.

The chosen mode of operation (bidirectional RRT with root heuristic) is shown to be slightly slower for the simpler maps than without the addition of the bidirectional RRT. However the time difference is small and outweighed by the benefits of having a mode of operation that can solve more maps in general. The position and number of obstacles is also shown to have a significant impact on algorithm run time.

Future work could investigate replacing the RRT entirely with an algorithm that solely uses Bezier curves, such as a genetic algorithm. Past work [13] would suggest such an option is feasible, however the problem of quickly determining which push points are accessible would need to be addressed. The use of Bezier curves could also be completely replaced with a full Reeds Shepps curve solution or extensions thereof similar to the work of Fraichard and Scheuer [22].

Finally the most obvious avenue for future work would be to conduct practical experiments using the algorithm developed in this research. This would involve many additional components to path planning, such as trajectory tracking [11], mapping and sensor integration and actuation and feedback control.

ACKNOWLEDGMENT

The authors would like to thank Marcus Swift for providing advice and guidance on his past research. Finally we would like to thank our friends and family, we are grateful for their support, particularly during the COVID-19 pandemic.

REFERENCES

- [1] M. Hirayama, J. Guivant, J. Katupitiya, and M. Whitty, "Path planning for autonomous bulldozers," *Mechatronics*, vol. 58, pp. 20–38, 2019.
- [2] M. Swift, "Push planning for continuous Sokoban problems," B.E. Thesis, University of New South Wales, June 2019.
- [3] D. Nieuwenhuisen, A. F. van der Stappen, and M. H. Overmars, "Pushing a disk using compliance," *IEEE Transactions on Robotics*, vol. 23, no. 3, pp. 431 – 442, 2007.
- [4] M. de Berg and D. H. Gerrits, "Computing push plans for disk-shaped robots," in *2010 IEEE International Conference on Robotics and Automation*, Anchorage, Alaska, USA, May 2010, pp. 4487 – 4491.
- [5] O. Ben-Shahar and E. Rivlin, "To push or not to push: On the rearrangement of movable objects by a mobile robot," *IEEE Transactions on Systems, Man and Cybernetics*

- *Part B: Cybernetics*, vol. 28, no. 5, pp. 667 – 679, October 1998.
- [6] K. M. Lynch and M. T. Mason, “Stable pushing: Mechanics, controllability, and planning,” *The International Journal of Robotics Research*, vol. 15, no. 6, pp. 533 – 556, December 1996.
- [7] T. Meriçli, M. Veloso, and H. L. Akin, “Push-manipulation of complex passive mobile objects using experimentally acquired motion models,” *Auton Robot*, vol. 38, pp. 317–329, 2015.
- [8] S. M. LaValle and J. Kuffner, Jr, “Randomized kinodynamic planning,” *The International Journal of Robotics Research*, vol. 20, no. 5, pp. 378–400, May 2001.
- [9] O. Salzman and D. Halperin, “Asymptotically near-optimal RRT for fast, high-quality motion planning,” *IEEE Transactions on Robotics*, vol. 32, no. 3, pp. 473–483, June 2016.
- [10] Q. dan Zhu, Y. bin Wu, G. qiang Wu, and X. Wang, “An improved anytime RRTs algorithm,” in *International Conference on Artificial Intelligence and Computational Intelligence*, 2009, pp. 268–272.
- [11] J. Shi, D. Sun, D. Qin, M. Hu, Y. Kan, K. Ma, and R. Chen, “Planning the trajectory of an autonomous wheel loader and tracking its trajectory via adaptive model predictive control,” *Robotics and Autonomous Systems*, vol. 131, 2020.
- [12] J.-W. Choi, R. Curry, and G. Elkaim, “Piecewise bezier curves path planning with continuous curvature constraint for autonomous driving,” *Lecture notes in Electrical Engineering*, vol. 10, pp. 31–45, 2010.
- [13] D. G. Macharet, A. A. Neto, and M. F. M. Campos, “Feasible uav path planning using genetic algorithms and bezier curves,” *Lecture notes in computer science*, vol. 6404, pp. 223–232, 2010.
- [14] R. Cimurs, J. Hwang, and I. H. Suh, “Bezier curve-based smoothing for path planner with curvature constraint,” in *First IEEE International Conference on Robotic Computing*, 2017, pp. 241–248.
- [15] M. Swift, H. Jayakody, and M. Whitty, “Path planning for multi-object push problems in continuous domain,” *IFAC-PapersOnLine*, vol. 52, no. 30, pp. 161–166, 2019, 6th IFAC Conference on Sensing, Control and Automation Technologies for Agriculture AGRICONTROL 2019. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S2405896319324346>
- [16] S. LaValle, “Reeds-Shepp Curves,” [http : // planning.cs.uiuc.edu / node822.html](http://planning.cs.uiuc.edu/node822.html), April 2012, accessed: 2020-03-10.
- [17] M. Svenstrup, T. Bak, and H. J. Andersen, “Minimising Computational Complexity of the RRT Algorithm: A Practical Approach,” in *IEEE International Conference on Robotics and Automation*, Shanghai, China, 2011, pp. 5602–5607.
- [18] J. Behley, V. Steinhage, and A. B. Cremers, “Efficient Radius Neighbor Search in Three-dimensional Point Clouds,” in *IEEE International Conference on Robotics and Automation*, 2015, pp. 3625–3630.
- [19] Nominal Animal, “Closest points on two line segments,” [https : // math.stackexchange.com / questions / 846054 / closest-points-on-two-line-segments](https://math.stackexchange.com/questions/846054/closest-points-on-two-line-segments), accessed: 2020-03-28.
- [20] M. Kamermans, “A Primer on Bézier Curves,” <https://pomax.github.io/bezierinfo/>, 2011, accessed: 2020-04-08.
- [21] S. Yoon, D. Lee, J. Jung, and D. H. Shim, “Spline-based RRT* Using Piecewise Continuous Collision-checking Algorithm for Car-like Vehicles,” *Intelligent Robotic Systems*, vol. 90, pp. 537–549, 2018.
- [22] T. Fraichard and A. Scheuer, “From Reeds and Shepp’s to continuous-curvature paths,” *IEEE Transactions on Robotics*, vol. 20, pp. 1025–1035, 2004.

Elliott Smith completed this work as an undergraduate student in Mechatronic Engineering and Computer Science at UNSW Sydney, and now works for WSP Australia. His research interests are in optimisation and planning.

Hiranya Jayakody was a postdoctoral researcher at UNSW Sydney and is the co-founder and CTO of HEO Robotics. His research interests are in space robotics, control and in computer vision. He has conducted applied research in a variety of domains from space to microscopy to agriculture.

Mark Whitty is a Senior Lecturer at UNSW and an IEEE member who leads the Smart Robotic Viti-culture group and lectures in robotics. His research interests are in image processing in agriculture, yield estimation and path planning.