A specialised collision detector for grape vines

Scott Paulin, Tom Botterill, XiaoQi Chen, Richard Green
University of Canterbury, New Zealand
scott.paulin@pg.canterbury.ac.nz

Abstract

Efficient motion planning is important for robot arms so that they can work productively. Standard methods for motion planning are often slow in complex environments because the collision detectors they use are inefficient. By customising a collision detector for the environment a large speed-up can be obtained. We have implemented a specialised collision detector for grape vines to speed up path planning. It models vines using capsules and spheres which provide fast intersection tests. Spatial partitioning is performed using sweep and prune. Objects are bounded by spheres to further improve computation times. The specialised collision detector is used on our vine pruning robot and we compare its performance to the Flexible Collision Library.

1 Introduction

We are developing a robot to autonomously prune grape vines [Botterill *et al.*, 2015] at the University of Canterbury, as shown in Fig. 1. Vines are reconstructed using stereo vision [Botterill *et al.*, 2014] and cuts are found with an AI algorithm [Corbett-davies *et al.*, 2012]. Cuts are made using a UR5 [Universal Robots, 2015] six degree of freedom robot arm with a spinning mill end, as shown in Fig. 2. To make a cut the robot arm must move the mill end through the cane. We use a path planner to compute collision free paths for the robot arm to make cuts. We have previously found that state of the art path planners are slow for this problem [Paulin *et al.*, 2015].

It is common for more than 90% of path planning computation time to be spent in collision detection routines [Sánchez and Latombe, 2003]. These times can be lowered by reducing the number of collision detection calls required, or speeding up the collision detector. Previous work focusses on reducing the number of times the collision detector is called by specialising the path planner to the robot's configuration space [Bohlin and Kavraki, 2001; Kuffner and Lavalle, 2000]. This is difficult to do with high degree of freedom robot arms because the configuration space cannot easily be visualised, however, it is possible to visualise the robot's environment. This means we can specialise the robot's collision detector to the task of vine pruning more easily than we can specialise a state of the art general path planner.

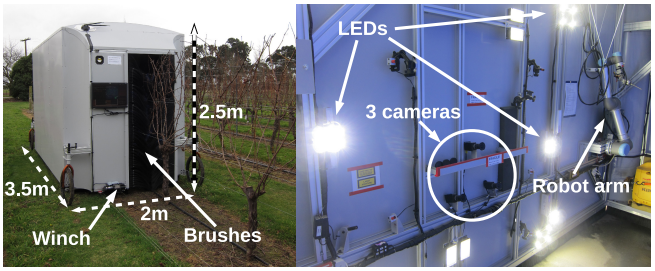


Figure 1: Vine pruning robot. Image courtesy of Botterill et al. [Botterill *et al.*, 2015].

Figure 2: Robot arm for pruning vines. Image courtesy of Botterill et al. [Botterill *et al.*, 2015].

2 Background

The collision detector must keep a model of the robot and its environment. Objects are commonly represented as a polygon soup (e.g. from a mesh), a solid primitive

(e.g. sphere, cylinder) or a combination of primitives. Ideally an objects representation will geometrically be a good fit and allow for fast collision checking.

A naive collision detector would perform all pairwise collision checks between objects in the environment. For our robot it would check every object in the robot arm against every object in the environment. This is slow when there is a large number of objects in the environment or robot arm. Most of the pairwise checks can be avoided by using spatial partitioning and bounding volumes.

One bounding volume can contain one or more objects. When encapsulating multiple objects the bounding volume can reduce the number of pairwise checks. We can also encapsulate objects that are computationally expensive to check, e.g. complex polygon soups, with objects that are simple to check like spheres. It is favourable to use bounding volumes that are a tight fit and provide inexpensive intersection tests, however, there is often a trade-off between the two. Some common bounding volumes (in increasing order of complexity) are sphere, axis aligned bounding box [Bergen, 1997], oriented bounding box [Gottschalk *et al.*, 1996], discrete oriented polytope [Klosowski *et al.*, 1998] and convex hull.

Spatial partitioning methods divide the space into regions and test whether objects overlap in the same region of space [Ericson, 2004]. Three common methods are grids, trees and sweep and prune [Cohen *et al.*, 1995].

Grid methods overlay space with cells. Pairwise checks are performed between the robot and objects in the same cells. The performance of these methods is highly dependant on the grids resolution. If it is too fine then many objects will be members of multiple cells. If the resolution is too course then there will be many objects in each cell. Both cases lower the performance of grids [Ericson, 2004].

Tree methods recursively divide the space into cells. This is similar to the grid structure, however, each cell is further split into cells until a certain tree depth or resolution on the leaf node is reached. Octrees are a common method. They recursively divide the space into eight cubes. This continues until a maximum tree depth is reached or the leaf node volumes are smaller than a specified value.

Grids and trees work well when environment objects can be tightly bounded by cubes. Long and thin objects, such as cylinders, are not tightly bounded because all dimensions of the cube scale with the largest dimension of the cylinder, its length. Cylinders will be loosely bounded by one cube, or occupy multiple partitions. Both of these cases cause a reduction in performance [Ericson, 2004].

Instead of grouping objects by the cubes they occupy, we can maintain them in a spatially sorted list and use sweep and prune[David Baraff, 1992; Cohen *et al.*, 1995]. The minimum and maximum distances between each object and a reference point, e.g. the base of the robot, are computed and stored. Sweep and prune finds a set of object pairs that cannot possibly be in collision based on these distances, as shown in Fig. 3. Intersection tests are then performed on the remaining pairs.

Safety margins can be applied to objects in the collision detector to account for sensor error. This allows a path planner to find paths with a guaranteed minimum obstacle clearance without requiring the collision detector to perform expensive distance-to-nearest-obstacle queries. These margins are often applied uniformly to all objects in the scene.

3 Specialising a collision detector for grape vines

A grape vine is made of canes and a head model. We model the canes with capsules (cylinders with hemispherical end caps) and the head model with a number of spheres, as shown in Fig. 4. We chose these primitives because they are quick to perform intersection tests on and are a good fit for our problem. The capsules are approximated as the union of a sphere and a ray to reduce intersection testing time.

Each capsule is bounded by a sphere because sphere-sphere collision checks are faster than capsule-capsule checks. The entire head model is bounded by one sphere to reduce the number of pair-wise collision checks. The robot frame and generator are modelled with planes.

To further reduce the number of pairwise checks we use a one dimensional sweep and prune algorithm (Fig. 3) similar to that in I-Collide [Cohen *et al.*, 1995]. The minimum and maximum distances to the base of the robot are computed for each object. Collision detection is only performed on pairs of objects that have overlapping minimum and maximum distances to the base of the robot. For example, to check for collisions between a capsule on the robot arm and a capsule on the vine the following would be performed:

1. Check whether the minimum and maximum distances to the robot base of the two capsules overlap. If they do, then the capsules may intersect and we should proceed to 2.
2. Check whether the bounding spheres for the two capsules intersect. If they do then the capsules may intersect and we should proceed to 3.
3. Perform an intersection test on the two capsules.

To make a cut the robot arm needs to move very close to the cane to make a cut. This means that any safety margins applied to the cane need to be small near parts of the robot arm when it is in a cutting configuration,

1: **procedure** SweepAndPrune(a, b) ▷ Determine whether a pair of objects cannot intersect
2: **if** (a.min() > b.max()) or (b.max() < a.min()) **then return** cannot_intersect
3: **else return** may_intersect
4: **end if**
5: **end procedure**

Figure 3: One dimensional sweep and prune. min() and max() provide the minimum and maximum distances between an object and a reference point e.g. the base of the robot. If there is no overlap of min and max values of the two objects then the pair cannot intersect.

but can be large further away. We add a safety margin to each object based on the minimum distance between it and any part of the robot arm when it is in a configuration to make any of the cuts on the vine, as shown in Fig. 4. Larger margins are added to objects further from the arm.

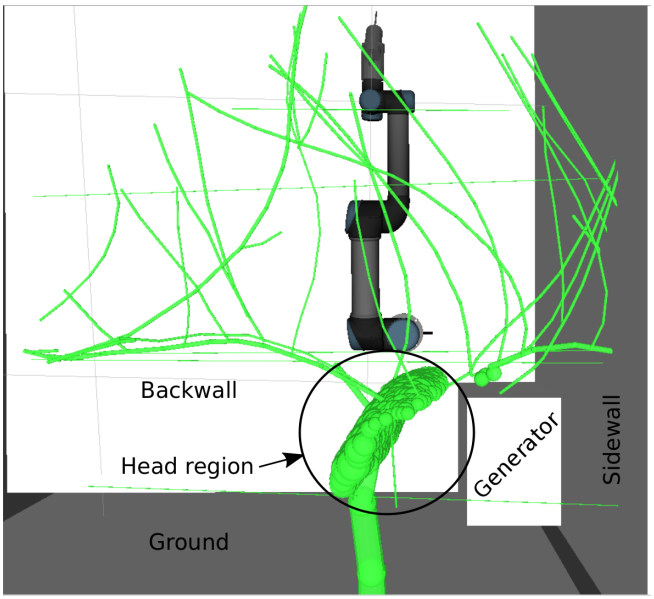
4 Results

We compare the specialised collision detector to the Flexible Collision Library (FCL) [Pan *et al.*, 2012] through Moveit [Chitta *et al.*, 2012]. FCL is configured by Moveit to use octree spatial partitioning and bounding volumes, as shown in Tab. 1. The robot arm is represented with a mesh provided by the Universal Robot package [Edwards *et al.*,].

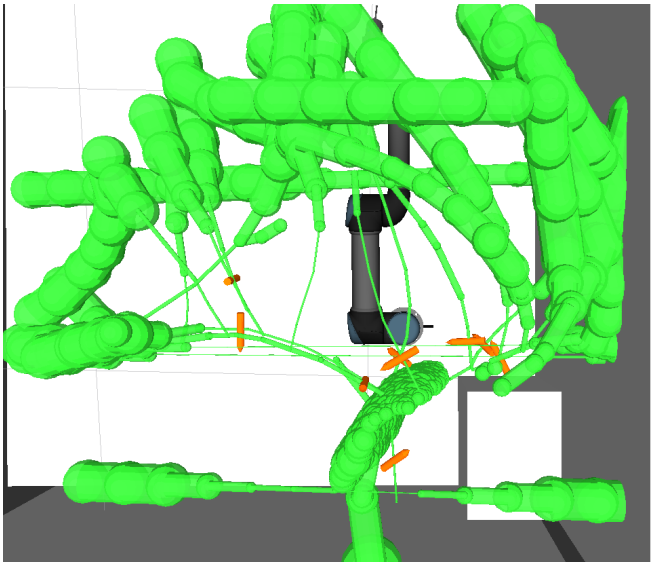
The specialised collision detector has significant speed-up over FCL for full collision checks, as shown in Tab. 2, and for self collision checks in Tab. 3. To compute mean times we generated a large number of configurations and timed both collision detectors on checking all states. The mean was computed from total collision checking time divided by the number of states checked. We performed trials with states that were randomly generated and with states which we knew to be valid. We tested using valid states because the collision detection algorithm cannot terminate early meaning it will often take longer, when testing an invalid state it can terminate as soon as a collision is detected.

Tab. 4 shows that the sweep and prune algorithm saves over 90% of pairwise object checks between the robot arm and the vine. These checks now only take a small portion of total collision detection time, as shown in Fig. 5.

Using the specialised collision detector results in the speed up in path planning times shown in Tab. 5. The path planner was tested by calculating all of the paths required to prune 37 plants twice each, which is approximately 650 paths. Different paths are generated each time a specific plant is pruned due to randomness in the cut point positioning and path planning algorithms.



(a) Grape vine without safety margins applied



(b) Grape vine with safety margins. Cuts are marked in orange. Cuts are made using specific robot states. The safety margin is computed using the minimum distances between parts of the robot arm in these states and the parts of the vine.

Figure 4: Adaptive safety margin applied to a vine

Table 1: Configurations of specialised collision detector and FCL

Collision detector	Arm model	Spatial partitioning	Uses bounding volumes
Specialised	Capsules	Sweep and prune	yes
FCL	Mesh	Octree	yes

Table 2: Mean times for full collision checking with smart safety margin

Collision detector	Random state [s]	Valid state [s]
Specialised	3×10^{-6}	5.9×10^{-6}
FCL	1.5×10^{-4}	2.5×10^{-4}
Improvement	38×	42×

Table 3: Mean times for self collision checking with smart safety margin

Collision detector	Random state [s]	Valid state [s]
Specialised	2.8×10^{-6}	2.9×10^{-6}
FCL	6.0×10^{-5}	5.0×10^{-5}
Improvement	21×	17×

Table 4: How the specialised collision detector identifies that pairs of objects are not intersecting when the input robot state is not in collision. Most objects pairs are classified as not intersecting by the sweep and prune algorithm. Intersection testing is only required for 0.23% of pairs on average.

	Mean [%]
Sweep and prune	91
Head bounding volume	6.2
Capsule bounding volumes	2.1
Pairwise checks performed	0.23

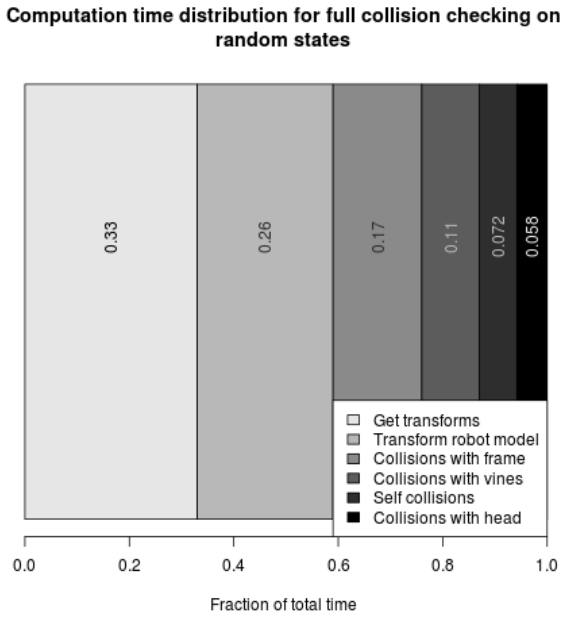


Figure 5: Breakdown of computation time for specialised collision detector. Most of the computation time is spent getting and applying transforms to the robot model rather than testing it for collision.

Table 5: Mean path planning and execution times when using the specialised collision detector. Execution time is how long it takes the UR5 robot arm to follow the planned path. For path planning with a 30 second time-out there was a 100% success rate for trials with the specialised collision detector, and a 99.3% success rate when using the Flexible Collision Library.

	With smart safety margin		Without smart safety margin	
Collision detector	Planning time [s]	Execution time [s]	Planning time [s]	Execution time [s]
Specialised with sweep and prune	0.13	6.5	0.12	6.7
Specialised no sweep and prune	0.51	6.5	0.43	6.7
Flexible Collision Library	-	-	3.4	6.1

5 Discussion

The specialised collision detector is fast because it performs few pairwise collision checks, as shown in Tab. 4. This is because the sweep and prune algorithm allows most pairwise object collision checks to be bypassed. Sweep and prune works well because most of the parts of the robot arm are closer to the robot's base than almost all of the obstacles for most robot states.

Putting a sphere bounding volume around the vines head region worked well because it was a tight fit and the head region had a large number of spheres. Without the bounding volume we would have always had to test against all spheres that were not removed by sweep and prune.

We bounded each capsule with a sphere because sphere intersection tests are cheaper than capsule intersection tests. This saved less pairwise checks than the bounding sphere around the head region because it only bounded one object. We could improve the collision detector by bounding multiple capsules with one volume e.g. by bounding an entire cane with one capsule.

Most of the collision detection time is spend getting the robot link transforms for the input state and applying these transforms to the robot collision model as shown in Fig. 5. This is because very few pairwise intersection tests are performed because of sweep and prune, as shown in Tab. 4. Performing intersection tests between the robot arm and robot frame, which only has six planes, takes about the same amount of time as testing the arm against the entire vine. This is because sweep and prune is not applied to these intersection tests.

Using the specialised collision detector in path planning provides a 26 times speed-up compared to using FCL, as shown in Tab. 5. Planning times with the specialised collision detector are also 12 times faster than those by Lee et al. [Lee *et al.*, 2014] who report a mean time of 1.5 seconds for a success rate of 80%.

Using adaptive safety margins allowed us to have a large safety margin on some parts of the vine as shown in Fig. 4. This provides a guaranteed minimum clearance between the robot arm and parts of the vine away from cuts in computed paths. Using the safety margin caused a small increase in path planning times (Tab. 5) because the vine became larger. The safety margin relies on having an accurate model of the vines around cuts. The real robot still collides with vines when the 3D model is inaccurate near cuts because the adaptive safety margin is small in those places.

6 Conclusion

Efficient motion planning is important for robot arms so they can work productively. Standard methods for motion planning are often slow in complex environments because the collision detectors they use are inefficient. By customising a collision detector for the environment a large speed-up can be obtained. We have implemented a specialised collision detector for grape vines. This specialised collision detector performs well on grape vines because it performs very few pairwise intersection tests. The pairwise tests it does perform are fast because the grape vines and robot arm are modelled with capsules are spheres which provide fast intersection tests. Path planning with our efficient collision detector means that planning times are small compared to execution times. The ability to compute large numbers of paths efficiently will allow us to find the best order of cuts to reduce execution times in future work.

References

[Bergen, 1997] Gino Van Den Bergen. Efficient Collision Detection of Complex Deformable Models using AABB Trees. *Journal of Graphics Tools*, 2(4):1–13, 1997.

[Bohlin and Kavraki, 2001] R Bohlin and LE Kavraki. A Randomized Approach to Robot Path Planning Based on Lazy Evaluation. *Combinatorial Optimization*, pages 221–249, 2001.

[Botterill *et al.*, 2014] Tom Botterill, Richard Green, and Steven Mills. A decision-theoretic formulation for sparse stereo correspondence problems. In *3DV*, 2014.

[Botterill *et al.*, 2015] Tom Botterill, Scott Paulin, Richard Green, Samuel Williams, Jessica Lin, Valarie

Saxton, Steven Mills, XiaoQi Chen, and Sam Corbett-Davies. A robot system for pruning grape vines. *Submitted to Journal of Field Robotics*, 2015.

[Chitta *et al.*, 2012] S Chitta, I Sucan, and S Cousins. Moveit! *IEEE Robotics Automation ...*, (March):18–19, 2012.

[Cohen *et al.*, 1995] Jonathan D Cohen, Ming C Lin, Dinesh Manocha, and Madhav Ponamgi. I-COLLIDE: An interactive and exact collision detection system for large-scale environments. In *I3D '95 Proceedings of the 1995 symposium on Interactive 3D graphics*, pages 189–197, 1995.

[Corbett-davies *et al.*, 2012] Sam Corbett-davies, Tom Botterill, Richard Green, and Valerie Saxton. An expert system for automatically pruning vines. In *IVCNZ*, 2012.

[David Baraff, 1992] David Baraff. *Dynamic Simulation of non-penetrating rigid bodies*. Phd, Cornell, 1992.

[Edwards *et al.*,] Shaun Edwards, Stuart Glaser, Kelsey Hawkins, Wim Meeussen, and Felix Messmer. universal_robot ROS package.

[Ericson, 2004] Christer Ericson. *Real time collision detection*. Elsevier, 2004.

[Gottschalk *et al.*, 1996] S Gottschalk, M C Lin, D Manocha, and Chapel Hill. OBB Tree: A Hierarchical Structure for Rapid Interference Detection. (8920219):171–180, 1996.

[Klosowski *et al.*, 1998] J Klosowski, M Held, Joseph S B Mitchell, K Zika\ N, and H Sowizral. Efficient Collision Detection Using Bounding Volume Hierarchies of k-DOPS. *IEEE Trans. Visualizat. Comput. Graph.*, 4(1):21–36, 1998.

[Kuffner and Lavalle, 2000] James J Kuffner and Steven M Lavalle. RRT-Connect : An Efficient Approach to Single-Query Path Planning. In *International Conference on Robotics and Automation*, number April, pages 995–1001, 2000.

[Lee *et al.*, 2014] James Ju Heon Lee, Kris Frey, Robert Fitch, and Salah Sukkareith. Fast Path Planning for Precision Weeding. In *Australasian Conference on Robotics and Automation*, 2014.

[Pan *et al.*, 2012] Jia Pan, Sachin Chitta, and Dinesh Manocha. FCL: A general purpose library for collision and proximity queries. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 3859–3866, 2012.

[Paulin *et al.*, 2015] S Paulin, T Botterill, J Lin, X Chen, and R Green. A comparison of sampling-based path planners for a grape vine pruning robot arm. In *International Conference on Automation, Robotics and Applications*, pages 98–103, 2015.

[Sánchez and Latombe, 2003] G Sánchez and JC Latombe. A single-query bi-directional probabilistic roadmap planner with lazy collision checking. In *International Symposium Robotics Research*, pages 403–414. 2003.

[Universal Robots, 2015] Universal Robots. Universal Robots UR5 robot arm, 2015.