A Novel Method For Fast Collision Detection on the PR2

William Marshall, *SUNFEST* – (CS), Lehigh University
Advisor: Dr. Camillo J. Taylor

Abstract

In the current robot motion planning pipeline for Willow Garage's PR2 robot, inefficient collision detection algorithms present a bottleneck. This paper provides background information, proposes a new, more efficient method for quickly determining whether a robot is in collision with its environment and gives a basic outline of the implementation process. The new method treats the PR2 robot as a collection of spheres that, given an array of joint angles, are positioned in the coordinate system of a depth camera. Whereas the old technique treats a depth map as a three-dimensional set of voxels that can be tested for collision using an octree, the new technique uses the idea that a depth map is a two-dimensional representation of free space from the perspective of the depth camera. If the robot is also rendered from the camera's perspective, and every point on each sphere that comprises the robot has a depth value lower than its corresponding point on the depth map, then the robot is not in collision with its environment. In addition, this technique can be applied to multiple camera perspectives of the same robot model. Finally, preliminary results and further possible optimizations are explored.

Table of Contents

1. INTRODUCTION

A current problem in robot motion planning is the speed and efficiency of collision detection algorithms. Collision detection denotes the set of techniques that are used to determine whether a particular robot pose is valid within a given environment. To understand why fast collision detection is important, one must first understand the basics of how the path of motion of a robot is computed.

The problem of trying to calculate a safe path for a robotic arm to take as it grasps a cup provides a good example. The degrees of motion of a robot's joint can be treated as a dimension in an *n* dimensional space, where *n* is the number of joints of the robot. Every potential pose the robot could take is then represented as a point in the robot's "joint space". A potential path that the robotic arm could take can be represented as a progression through a series of poses, or points, that have been shown to be valid [1]. Thus, in order to plan the motion of a robotic arm the robot's joint space must be sampled to distinguish the points which are valid from the points which collide with the environment.

The issue with this approach to motion planning is that it is impossible to be certain that the path a robot might take is valid, since there are an infinite number of poses that would need to be sampled. A partial solution to this problem, termed RRT planning, is to take a suitably high resolution random sampling of the joint space, such that it would be improbable for there to be a collision between any two neighboring points [2]. Given the high dimensionality of the joint space, it may be necessary to validate thousands or even millions of robot poses. Unless efficient methods are used, this process can become time consuming.

2. BACKGROUND

The previous method for collision detection on the PR2 robot was capable of checking the validity of approximately 65,000 poses per second. The speed can be attributed to the use of octrees. An octree is a data structure that allows for segmentation of 3 dimensional special regions based on whether they contain an obstacle, and allows for nlog(n) search time to determine whether a robot is in collision with its environment [3].

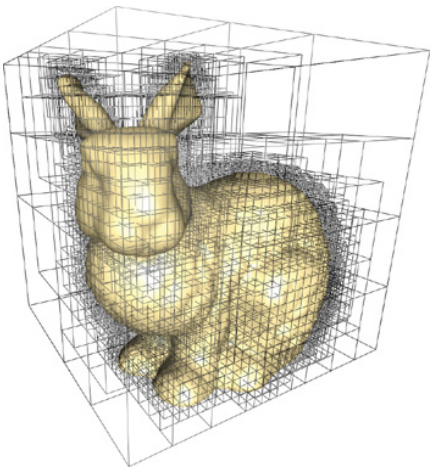

Fig 1: An octree segmenting a rabbit's space. **http://http.developer.nvidia.com/GPUGems2/elementLinks/37_octree_03.jpg**

Despite the efficiency of an octree, collision detection still proves to be a bottleneck in the robot motion planning pipeline; the PR2 robot needs approximately one second to calculate a path, the majority of which is spent sampling the robot's joint space. Therefore, if progress is to be made with robot motion planning, more efficient methods are warranted.

2.1 The PR2 Robot

The PR2 robot is a humanoid robot that was lent to the GRASP lab Willow Garage. It has over 80 separate joints, allowing for precise and complex manipulation of objects such as cups. In addition to force-feedback grippers, accelerometers, and stereo cameras the PR2 robot has a head mounted Microsoft Kinect depth camera that allows it to receive real time information about its environment [4].

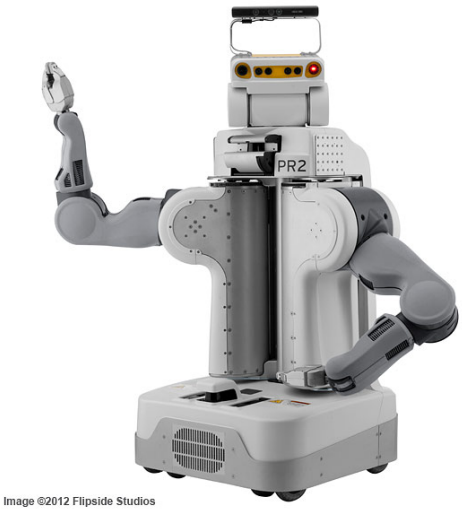

Fig 2. The PR2 robot.
http://flipsidestudios.com/wp-content/uploads/2012/04/pr2_web.jpg

3. The Fast Collision Detection Algorithm

The new method that we employ makes use of the idea that when a depth camera captures a scene it implies that the space between the camera and the surfaces it sees is free. If a robot is not in collision with its environment, then it will be completely contained within this free space. An easy way of checking to see whether a robot is contained in free space is to render a simulated robot into the scene from the perspective of the camera, and compare each of the robot's pixels with its projection onto the depth map.

Pseudo code for the algorithm would be as follows:

```
bool collisionChecker() {
        for each joint:
                for each sphere:
                        for each pixel:
                                if(pixel.depth>=depthMap.getDepthValue(pixel))
                                        return true;
                                else
                                        return false;
        }
```

If every pixel on the robot has a lower depth value than its corresponding point on the depth map then it is not in collision with its environment; conversely, if any pixel on the robot is found to have a depth value greater than its point on the depth map, then the robot would be in collision with its environment.

4. Robot Data Structure

This method is dependent on constructing a robot model that can determine the locations of the robot's joints in a real world coordinate system given the joint angles of the robot. The data structure of our model was based off of Willow Garage's Universal Robot Descriptor Format (URDF) which treats the PR2 robot as a tree of joints [5]. In addition to an axis which determines the direction of the joint's rotation, each joint is attributed a rigid body transformation that represents the rotation and translation from the coordinate system of the child joint to the parent joint. The rotation of a parent joint would then affect the transformations of every subsequent child joint deeper in the tree. Using the data structure, we can compute the real world coordinates of each of the PR2 robot's joints and render the robot into the scene.

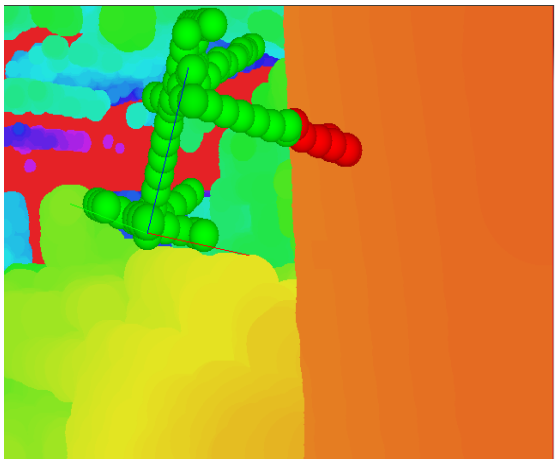

Fig 3. The PR2 robot rendered into the depth map of a Kinect.

5. Optimizations

Although the urdf format supports the use of dae and stl meshes as a means of accurately rendering the shape of the robot, our method instead uses spheres. Despite the loss of precision from using spheres instead of meshes, spheres allow for ways of optimizing the collision checker.

5.1 Scan Line Interval Tracking

One such optimization has been termed scan line interval tracking. Scan line interval tracking utilizes the idea that separately checking each sphere for collision is redundant, since different

spheres might occupy the same region of the depth map. If it is known that each sphere to be checked is closer to the camera than the previous one, then it would be impossible for a collision to occur in a previously tested region. While this optimization requires that each sphere in the robot model be ordered based on its depth value, the overhead of sorting a list is offset by the fact that the overlap between spheres need not be checked. The 'do not check' regions of the depth map are maintained by keeping a list of intervals for each scan line which represent the beginning and end of the sphere's projection onto the depth map (Fig 4). For models that consist of a large number of overlapping spheres this optimization is highly beneficial.

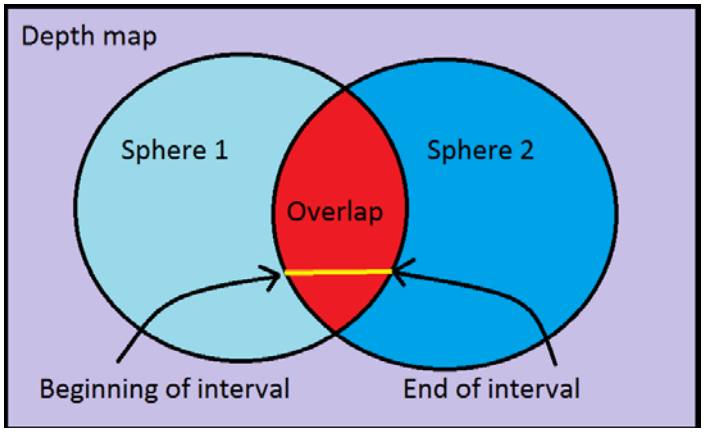

Fig 4. The overlap of the sphere's projections onto a depth map, and an example scan line interval

5.2 Bloom Depth

Another benefit that results from using uniformly sized spheres to represent our model is the 'bloom depth' method for removing some of the collision detector's false negatives and reducing the per-sphere testing from linear to constant time. Since the usage of spheres provides only an approximation of the robot's shape, our model cannot guarantee a conservative answer; certain robot poses might be validated despite being in collision. Although this flaw seems to imply that this collision detection method is unreliable, the errors would mostly occur when the robot performs complex maneuvers (such as cup grasping) and is less of an issue when the robot has less proximity to its environment. In addition, the bloom depth method reduces this uncertainty by altering the depth map such that each point is set to the minimum depth value within the radius of the sphere. As a result, rather than testing the corresponding depth values for every pixel on a sphere, any subsequent sphere checks only need to test against the value for the sphere's center.

5.3 Optimization Comparison

Between the two optimizations, bloom depth has the advantage of being both faster and more cautious with the result. Bloom depth also requires less memory since it doesn't have to maintain a list of intervals, and only requires 17.5 Mb of system memory to run. A disadvantage to using

bloom depth is that the requirement for uniform sphere size removes granularity, while scan line interval tracking allows the model to distinguish between different types of joints. Unfortunately, the methods are mutually exclusive, so it is necessary to decide between granularity and speed. Since our goal in this project was to speed up the collision detection bottleneck, bloom depth presents the better optimization.

6 Results

Running on a Core i5 2500k processor, our collision detection method operates at an average of 75,000 Hz for a model of the PR2 robot that was comprised of 209 spheres. For a mid-range processor, such as the Core i7 720QM, the rate slows to approximately 50,000 detections/sec. When additional robot joints are added, the detection time increases linearly, as shown in figure 5. It is important to note that the computation for validating a robot pose against a single depth map is done on only one of a processor's cores. In future implementations of this method it may be possible to utilize multiple cores to validate a robot's pose from more than one perspective, without adverse performance degradation.

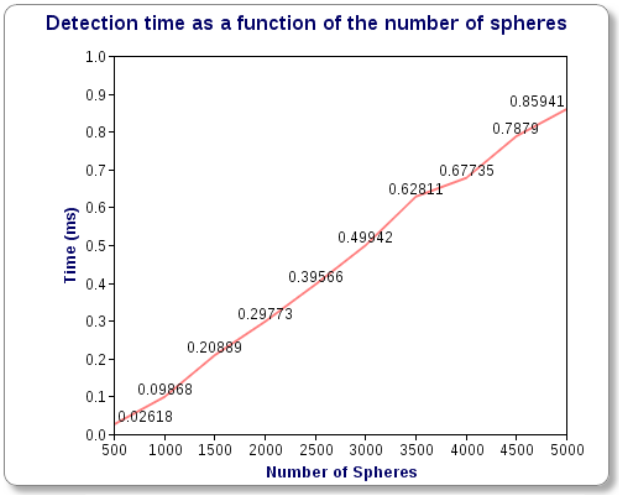

Fig 5. The relation between number of spheres and detection time

7. Future work

This collision detection method has the potential to be further parallelized on the GPU using OpenGL. In the OpenGL pipeline, the rasterization and fragment shader steps perform calculations on pixels in parallel on the GPU. When a pixel is sent to the fragment shader to be assigned a value from a texture map, it is sent independently from and concurrently with the other pixels being rendered [6]. This process is similar to what is performed in our collision detection method when each pixel of a sphere is assigned a value from a depth map. Instead of using a pure C++ implementation, future work will involve a parallel solution by utilizing OpenGL's rasterization process to implement the per-pixel testing, and OpenGL's texture maps to

store the depth values.

9. References

[1] Lydia E. Kavraki, Petr Svestka, Jean-Claude Latombe, and Mark H. Overmars, "Probabilistic Roadmaps for Path Planning in High-Dimensional Configuration Spaces," IEEE TRANSACTIONS ON ROBOTICS AND AUTOMATION, VOL. 12, NO. 4, 566-580, AUGUST 1996

[2] Kuffner, J.J., Jr.; LaValle, S.M.; , "RRT-connect: An efficient approach to single-query path planning ," *Robotics and Automation, 2000. Proceedings. ICRA '00. IEEE International Conference on* , vol.2, no., pp.995-1001 vol.2, 2000

[3] Madhav, Dinesh Manocha, and Ming C. Lin. 1995. "Incremental algorithms for collision detection between solid models," In *Proceedings of the third ACM symposium on Solid modeling and applications* (SMA '95), Chris Hoffmann and Jarek Rossignac (Eds.). ACM, New York, NY, USA, 293-304.

[4] Chitta, S.; Cohen, B.; Likhachev, M.; , "Planning for autonomous door opening with a mobile manipulator," *Robotics and Automation (ICRA), 2010 IEEE International Conference on* , vol., no., pp.1799-1806, 3-7 May 2010

[5] Kunze, L.; Roehm, T.; Beetz, M.; , "Towards semantic robot description languages," *Robotics and Automation (ICRA), 2011 IEEE International Conference on* , vol., no., pp.5589-5595, 9-13 May 2011

[6] Allard, J.; Raffin, B.; , "A shader-based parallel rendering framework," *Visualization, 2005. VIS 05. IEEE* , vol., no., pp. 127- 134, 23-28 Oct. 2005