Planning paths for a mobile robot with different upper and lower footprints

Arunkumar Byravan
Masters in Mechanical Engineering
University of Pennsylvania
Philadelphia, PA

Maxim Likhachev
School of Computer Science
University of Pennsylvania
Philadelphia, PA

Abstract**:** ***In this project, a simple method for efficient and real-time planning for a mobile robot with differently shaped upper and lower bodies is presented. The approach is based on planning paths, taking into account possible collisions of both the upper and lower parts of the robot. To generate fast plans, the 3D space is projected onto a set of 2D planes and these are used for planning and collision detection. Simulation results with the PR2 robot in a ROS setup and different environments are compared with those from current implementations.***

I. INTRODUCTION

Autonomous mobile robots are nowadays being fitted with manipulators to enable them to perform a wide variety of actions such as grasping and moving objects from one place to another. Such robots usually have a polygonal or a circular shaped base and a very different upper body with the arm. In these cases, planning for the whole robot becomes tougher. To make the planning problem easier, in most generic cases, the arms are assumed to be tucked into the robot and then planning is done just for the base by projecting the space onto a 2D plane. This approach will not always be feasible. There may be many actions that may not allow tucking the arms back into the robot. In such cases, the existing implementations would fail to detect the collisions of the arm. One way to go around this is to project the arms onto the ground plane as well and plan for the new footprint including the arms.

But this may not return any solutions as we are projecting obstacles that may not necessarily be in collision with the arms. So, we need to plan and check for collisions taking into account that the arms are at a certain height from the ground level.

The collision detection step has to be done for every single point along the plan and so, checking to see if every point is an obstacle, in 3D space would be very expensive and time-consuming. So, the dimensionality should also be reduced in some manner for the sake of efficiency.

The proposed method addresses both these issues. In addition to planning for the base, the arms are also projected onto separate planes and the footprints of the arms are checked for collisions against the projected obstacles. The main difference is that for the arms, only obstacles that are within its z-extents are considered when checking for collisions. Simulations were run in a ROS setup and different environments with the PR2 robot using an ARA* planner. The results show a marked improvement in detecting collisions compared to existing implementations.

II. RELATED WORK

The planning problem is usually formulated as a graph search. Graphs can be constructed in many ways as a Grid-based or a Lattice-based or as a Voronoi diagram, as a PRM etc. The states in a graph can just be representative of positions in the world or positions and

orientations or may be in the C-Space of a manipulator etc.

There are a number of established methods for searching these graphs, like Djikstra, A*, ARA*, D* Lite etc. Most of these give specific and proven bounds on the optimality of the path they return. They are also made to be very efficient and fast and usually are able to plan in real time.

The main bottleneck in the planning problem is checking for collisions. The speed of the planner depends on the speed of the collision checking part. A lot of research in collision detection has been related to its application in games for detecting collisions and planning suitable responses for animated characters. A number of techniques have been developed for fast detection in 2D as well as 3D such as the simple sweep and prune, finding the Separating Axis and checking for intersections using it etc. [4]. There are also a number of methods where a bounding box is fitted to the objects in question and broad phase collision detection is done to check if there is a chance for any collision between the objects. If the possibility of collision is high, the bounding boxes are then subdivided into smaller parts and each of them is checked for collisions and so on. [5] gives a good overview of current techniques for collision detection used in Computer Graphics.

These techniques have been translated to collision detection for robot manipulators. Usually, the checking is done in 3D using Bounding Boxes for the links, which turns out to be very expensive.

III. PROPOSED APPROACH

In a general sense, the proposed approach could be applied to any robot which does not have a uniform shape throughout its height. Initially, the robot is split into a number of distinct planes and all the obstacles within two planes are projected down onto the lower of the two planes. The 2D projection of the portion of the robot contained within the two planes, called the *footprint* of the robot, can then be used for checking collisions of the robot with any obstacles in that plane. The planning is done only on the lowermost plane with the whole robot and all the obstacles being projected onto that single plane. But, for every point along the plan, all the footprints are checked with the obstacles in their respective planes to detect collisions. A tolerance can be set which would determine the number of planes that the robot needs to be subdivided into. This can be set to different values based on the accuracy that is needed. A greater number of planes would take the approach closer to that of full blown 3D collision detection but would also take more time to process. On the other hand, a balance can be found between accuracy and speed which would dictate this tolerance.

IV. BACKGROUND

ROS has been chosen as the framework to work on due to its inbuilt features to generate maps from sensor data and the ease with which the solutions can be verified using tools like Gazebo and Rviz.

ROS has an existing global planner which makes use of the Djikstra's Algorithm and A* for planning. Also, a class of planning algorithms called the Search Based Planning Library (SBPL) are also implemented in ROS. All these planners used ROS's inbuilt Costmap generating package to get data about the environment. Also, none of the existing implementations include any collision checking for the arms. The planning and collision checking are done with respect to the base footprint and so the manipulators would usually hit the walls when the robot went too close to them. So, the focus was to implement the new method in ROS to prove the superiority of the new approach in comparison to the current existing implementation.

Due to the availability of data and models of the PR2 robot in ROS, we decided to implement the proposed system on the PR2 robot model in ROS. Fig 1 shows a model of the PR2 robot.

A few general assumptions were made related to the problem which included assuming perfect sensing, a static environment and no movement of the arms when the base of the robot was in motion. An ARA* planner is used for generating the plans and the search space is subdivided into a 3D (x,y,θ) lattice.

Fig 1. The PR2 robot, courtesy Willow Garage

V. IMPLEMENTATION IN ROS

The sections below deal with implementation of this proposed system on the PR2 robot in a simulated environment using ROS. Only two extra planes are considered, one for each of the arms.

The overall system can be thought of to be comprised of three separate parts, each of which will be explained in the sections below.

Footprint of the arms:

The PR2 has two 7-DOF robot arms. There are three main links which are about 0.4, 0.32 and 0.2 m in length. The links are assumed to be cylindrical. The procedure to obtain the footprint of the arm is simple. Initially, the joint angles are obtained from the ROS parameter server. Details about the joints, links and their limits can be used to create a simple model of the robot arm. These are available as pre-existing ROS files. Using these files and the current joint-angles, we compute the position of each joint in 3D space using Forward Kinematics.

Simply neglecting the "z" coordinates of these points will give us the projections of the joint centres on the x-y plane. These projections will be points along lines drawn through the centre of each of the links. We then find the perpendiculars to these lines to find the projections of the cylinders (links assumed to be cylinders). This gives us the footprint of the arm. This procedure can be repeated for the other arm as well. Figures 2 & 3 shows the footprints of the arms for two simple cases.

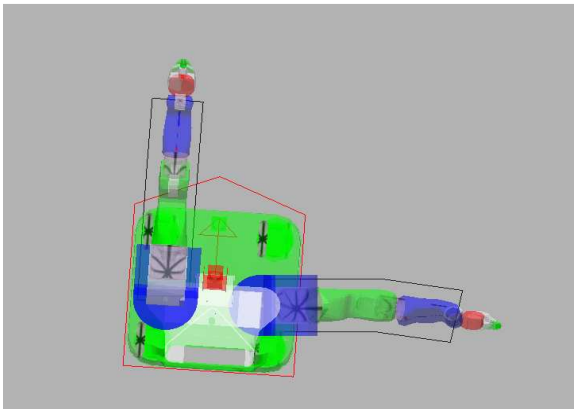

Fig 2 -- Footprints of the arm (in black) without considering the gripper (base footprint set manually – in red)

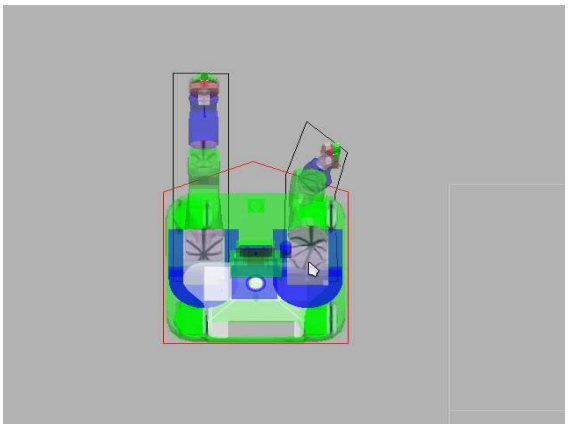

Fig 3 -- Footprints considering the gripper

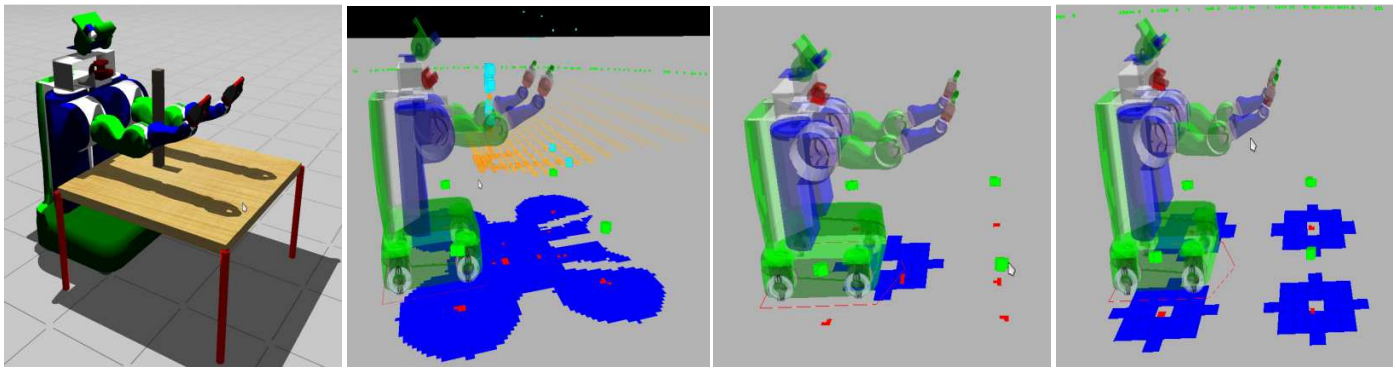

Fig 4: (From left to right – Blue represents inflated obstacles) a) PR2 robot in an environment with a table and a stick in between the arms, b) Costmap generated for the base, c) Costmap for the arms d) Costmap generated with sensor info from "Z" values of 0 to 0.4 m.

Costmaps for the arms:

The PR2 has a number of sensors namely a tilt laser scanner above the shoulders, two stereo cameras in its head and another laser scanner in its base. All the sensors output a point cloud which is the input to the Costmap. The Costmap projects all these points onto the XY plane and for every point, a cost is assigned based on the distance to the nearest obstacle. So, the costmaps we get are a 2D projection of the 3D world. Also, the obstacles are inflated by the inscribed radius of the robot. This helps in fast collision checking.

After we find the joint positions using Forward Kinematics, we compute the "Z" extents of the robot arm. The "Z" extents are the maximum and minimum Z values of the arms. The focus now is to get a map which projects obstacles only within this specified "Z" range. We accomplish this by filtering the sensor data that the costmap gets so that no data outside the necessary "Z" range is given to the costmap. We do this by changing a few parameters that are specific to the sensor data in the costmap. This map is now used for collision checking of the arm. Figure 4 shows a few costmaps that are obtained for different "Z" ranges from a specific environment using the procedure described above.

Collision checking and planning:

Using these new costmaps and footprints, we check for collisions of the arms as well as for the base in the planner. The current system (the ARA* planner in the (x,y,θ) lattice environment) checks for collisions of the base alone when planning. The planner is mainly based on the one described in [1]. It also uses a number of optimization methods described in [1] to speed up the planning process.

At the very first initialization step, the planner pre-computes all its actions using the possible motion primitives of the robot. It also pre-computes the footprint cells for all the actions. This enables it to plan very quickly and in real time.

In the proposed method, the planner still plans just for the base. But for every point in the planned path, the planner evaluates the footprints of the arms and checks to see if they collide with any obstacles. The new planner also pre-computes the necessary actions and footprints right at the start to improve the speed and performance.

The collision checking is done in two phases. There is a simple broad phase collision detection which initially checks to see if there are any obstacles within the inscribed radius

of the robot/footprint. If it detects any such cell, it definitely knows that the robot is in collision. It no such collision is detected, it checks to see if there are any obstacle cells within the circumscribed radius of the robot/footprint. If there are any cells detected here, it goes to the narrow phase collision detection which iterates through the cells in the footprint to look for collisions. This design helps the system to check for collisions in an efficient manner.

VI. EXPERIMENTAL RESULTS

We have implemented our system on a ROS setup using the PR2 robot model available in ROS. A number of simulations were run and ROS's visualization tools Gazebo and Rviz were used to check the feasibility of the obtained solution. The planner was tested out on two different environments, a simple maze like world with walls and no external obstacles and another world shown in Fig 3 (leftmost image). The results were compared with those obtained using the existing implementations. The results can be summarized as to belong to one of the following cases:

1) If both the arm and the base would not collide, both the planners returned the same solution. *(except for a few special cases tested)*

2) In the cases where the arm would collide but the base would not collide with any of the obstacles, the proposed planner indicated that there was no solution possible while the existing planner returned a solution which would cause the arms to collide with the obstacles.

3) In cases where both the arm and the base would collide, both planners indicated that no solution was possible.

As indicated above, a few special cases were tried out where initially both the base and the arms were not in collision but a simple motion would cause the arms to collide with obstacles. A simple case tried out was to take the start pose of the robot to be the one in Figure 3. The goal for that case was set such that the robot would end up facing the opposite direction about a meter behind its initial position. For that specific case, the existing planner returned a solution which asked the robot to spin in place and then move towards the goal. This ends up causing the arms to hit the stick in between them.

On the other hand, when the new planner is given the same case, it commanded the robot to go in reverse at first and then turn at a point where the arms are out of collision of the stick. This case proves the marked improvement of the proposed approach in comparison to the existing planner.

Table 1 shows the planning time for the proposed planner with respect to the size of the path. This shows that the new system is capable of planning paths in real time.

Path size (no of points) (1 point ~ 50 cm)	Proposed planner plan time (sec)
17	0.05
20	0.06
28	0.30

Table 1: Planning time for paths of different sizes.

Fig 5 shows another case used for testing the planner. The final goal position set is invalid due to collision of the arm. The existing planner returns a path which results in collision while the proposed planner doesn't return a path.

VII. CONCLUSIONS

We have presented a general approach for planning paths for robots with complex

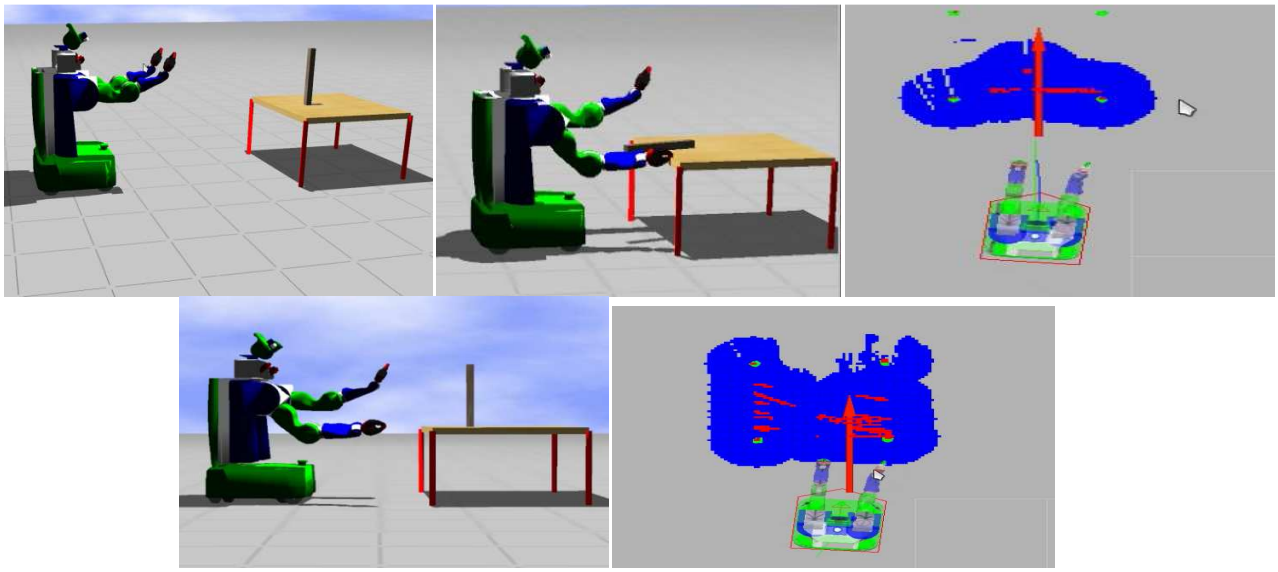

Fig 5 - (Going Clockwise from top) a) Initial pose of the robot b) Collision of arm with the table in case of the existing planner. c) Existing Planner shows that there is a path(in green) while there isn't one d) New planner doesn't allow the goal state to be set e) New planner shows no path

shaped bodies. Our approach uses simple projection of the 3D world onto the 2D space to generate safe collision free paths in real time. We also presented a method for generating the maps and footprints which are integral in our approach. A simple implementation of the proposed approach was implemented in ROS and a number of simulations were run and verified using the PR2 robot. The efficiency and speed of the proposed planner were compared with the existing method and found to be almost the same. The proposed method was also found to give better and collision free paths in most of the cases.

In the future, it is important to look at various ways to improve on the results obtained and to develop the proposed approach for more complicated cases. The implementation can be extended to check for collisions in multiple planes to achieve a better accuracy in detecting and avoiding collisions. The approach can also be extended to include motion of the arm as well. A challenge in this would be to be able to generate costmaps very quickly. Another problem would be that just projecting the obstacles from the top might not be enough in some cases. The approach can be improved to incorporate methods such as a finding the separating plane between the two objects and projecting them onto that specific plane to check for intersection.

REFERENCES

[1] Maxim Likhachev and Dave Ferguson, *"Planning Long Dynamically-Feasible Maneuvers for Autonomous Vehicles,"* International Journal of Robotics Research (IJRR), Vol. 28, No. 8, 933-945 (2009)

[2] Benjamin Cohen, Sachin Chitta, and Maxim Likhachev, *"Search-based Planning for Manipulation with Motion Primitives,"* Proceedings of the IEEE International Conference on Robotics and Automation (ICRA), 2010 (to appear)

[3] Maxim Likhachev, Geoff Gordon and Sebastian Thrun, *" ARA*: Anytime A* with Provable Bounds on Sub-Optimality,"* Advances in Neural Information Processing Systems 16 (NIPS), MIT Press, Cambridge, MA, 2004

[4] *"Rigid Body Simulation"*, David Baraff, Pixar Animation Studios

[5] "*Physics-Based Animation*", Erleben et al, 2005

[6] *www.ros.org* -- ROS Documentation