Merging Path Planners and Controllers through Local Context

Sundar Narasimhan
MIT Artificial Intelligence Laboratory
545 Technology Square, NE43-826
Cambridge, MA 02139
sundar@ai.mit.edu

Abstract

This paper presents an implemented approach to robotic tasks involving intermittent contact and changing dynamics in uncertain environments. The approach is to use global planning to find paths in a tesselated representation of the environment, and a set of local controllers to take into account possibly time varying dynamics. The important difference from conventional path-planning in robotic tasks is how this approach uses local sensory information, and the important difference from reactive or behavior-based approaches is that the local controllers are learnt from simulation models or actual trials and are not programmed in a-priori.

1 Introduction

Consider a scene as shown in Figure 1. This scene shows a number of polygonal objects resting on the plane. Also shown is a rod (which is a simplified form of a robot) which can move in the plane. While so doing, this rod contacts and pushes the objects in the environment. To simplify what follows, we will assume that the objects in the environment are fixed and only one of them beside the robot can move in the environment. The task is, stated very simply, to move a specified shape from a starting configuration s_i to a specified ending configuration s_g.

This task looks simple. Yet it shares a number of important features with other tasks:

1. It requires interacting with the environment, especially with the object the robot intends to move. In this particular task, because of the lack of a gripper, contact is intermittent.
2. The environment is filled with uncertainty. The positions of the objects are known only to within a certain accuracy, and the effect of actions cannot be predicted with arbitrary precision. More importantly, the pressure distribution at the object's surface in contact with the table underneath is unknown.
3. While moving the object in its environment, the robot and the moving object can come into contact with obstacles in the environment. The dynamics of predicting the evolution of the object's state through such contact situations is difficult if not impossible (see Mason and Wang [1988], Erdmann [1984] and more recently Baraff [1993] about inconsistencies that can arise when one tries to apply models of Newtonian mechanics along with the Coulomb model of friction).

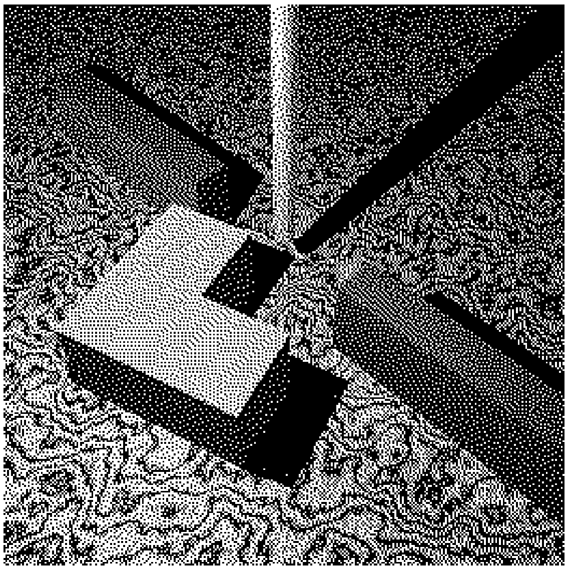

Figure 1: The Robot and its Environment

2 Previous Work

Given the nature of this problem, how can one program a robot to carry out this task? There have been many approaches to this fundamental problem of a robot's interaction with its uncertain environment. However, one can classify almost all of these into two broad categories. In what follows, we will instantiate the two approaches in our pushing domain, in order to illustrate their main characteristics.

The first approach relies on a rather clean separation of a *planning* component from the *execution* unit. In this approach, a geometric model of the environment is acquired and a path for the object to be pushed is planned. A highly abstract description of an example program for such a task might look like:

```
environment = sense
path = findpath (object, s-i, s-g, environment)
push-object (object, path)
```

The last line in the above program suggests where the execution component is utilized. This execution unit is saddled with the task of actually pushing the object along the path in an uncertain and possibly changing environment.

This basic idea has been explored for a wide variety of robot tasks. By separating the planning problem from execution, one could optimize and study them independently. The output of the planner is a sequence of actions. Its input comes from the sensors. However, the raw sensor values are usually pre-processed and merged into a world model.

In contrast to this approach stands the *reactive* and *behavior-based* approach for programming robot tasks[1]. The behavior-based technique was originally suggested by Brooks [1986] and has been elaborated by a number of researchers (see Connell [1990] and Brooks [1991]). To program a robot to push an object, one would hand-craft a series of behaviors that map sensory values to actions. Each of these behaviors can be thought of as finite-state machines in the behavior-based approaches or as simple table-lookups in purely reactive systems. Many such behaviors are hooked up in a topology pre-determined by the programmer. Mataric [1990] presents an example network for the navigation task, and Brock [1993] presents a network for the grasping task.

Conventional planners are brittle in the sense that they cannot handle uncertain or changing environments. They also have to face the problem of sensor fusion in order to merge different sensing inputs into a common world model. Reactive and behavior-based systems on the other hand require careful hand-crafting and anticipation, at design time, of the consequences of entire sequences of actions. These disadvantages have resulted in hybrid architectures that attempt to merge planners with reactive systems.

3 Approach

As has been recognized by many researchers, a framework that provides for both global geometric planning and local sensory interaction in order to take uncertain dynamics into account would be extremely useful. In what follows, we describe our approach to tackle this problem and illustrate how it works to solve problems in the pushing domain. Our approach is a hybrid one that has planning and reactive components. While we have explored issues related to the retention of state in our reactive component, this paper does not present those results.

In our pushing domain, the state of an object can be parametrized by three variables $s = (x, y, \theta)$, and consequently the configuration space (see Lozano-Perez [1983]) of the moving object in its environment is three dimensional. Our model for the dynamics will be quasi-static because it allows the *state* space to be identified with the configuration space of the moving object. It should be noted that our model for $\mathcal{S}$ is infinite (i.e. $s \in \mathcal{S} = \mathcal{R}^2 \times \mathcal{SO}^1$).

Our model for an action will be a tuple $A_i = (x, y, dx, dy)$ where the first two numbers quantify a point on the pushed object where the robot starts to push the moving object, and the last two specify the length and direction of the push in a co-ordinate system fixed on the moving object. This model of actions is object-centered and comprises the set of linear one-step pushing actions. It should be noted that our model for $\mathcal{A}$ is also infinite.

Our approach will be to use a geometric planner to plan feasible paths for the pushed object. An execution unit that attempts to push the object along this path will be presented. Both these components are assumed to be running in parallel. The information that is communicated from the planner to the execution unit is the set of geometric parameters that specifies the next segment along a path to the goal configuration.

We rely on feedback loops to ensure disturbance rejection and robustness. We will present a controller that can deal with the varying pressure distributions at the support surface quite effectively, when the environment contains no obstacles. To deal with the fact that the dynamics of pushed objects can change quite drastically upon contacting these obstacles, we present an approach that relies on switching controllers based on sensed local configurations.

4 Global Path Planning

We assume that a model of the environment exists and that a global path planner operates on this environment to produce a path. All our objects are assumed to be polygonal. In this paper, we do not address issues of how such a environment model can be built from sensory input or how complete it needs to be[2].

We will also assume that although many objects populate the environment at any given time, the robot's task is to move only *one* object from a specified starting configuration to a specified goal configuration.

The configuration space of the environment characterizes all legal positions for the moving object where no collisions exist between the moving object and the stationary obstacles. Obstacles in the configuration space consist of configurations where the moving object would collide with one or more of the stationary obstacles. Our planner constructs these configuration space obstacles using a modified version of the algorithm first outlined by Guibas, et al. [1983]. We then search a discretized representation of this configuration space to find paths connecting start and goal configurations.

Two views of the constructed configuration space for the example in Figure 1 are shown in Figure 2.

Paths found in the tesselated configuration space can be displayed graphically. In Figure 3 we show two such paths found by the planner, the second of which contains many more segments involving rotations than the first.

The path-planner's output is encoded by a series of configurations $P = x_i, y_i, \theta_i$.

5 Local Control

Even though we now have a desired path for the pushed object, we have not yet specified how the robot is to effect the motion of a pushed object along this specified path.

We have explored three alternatives for specifying local controllers.

1. The first approach hand-codes specific behaviors intended to effect particular object-relative motion. For example, we wrote behaviors tuned to translate the object along the path, and other behaviors that sought to rotate the object clock-wise or counter clock-wise. Although some effort was spent on making these behaviors not be specific to the shape of the

[1] We acknowledge the distinction between purely *reactive* and *behavior-based* systems. The latter retain state, can use internal representations and are often implemented on distributed systems.

[2] In our implementation, we use a camera mounted beneath the environment which gives us this information.

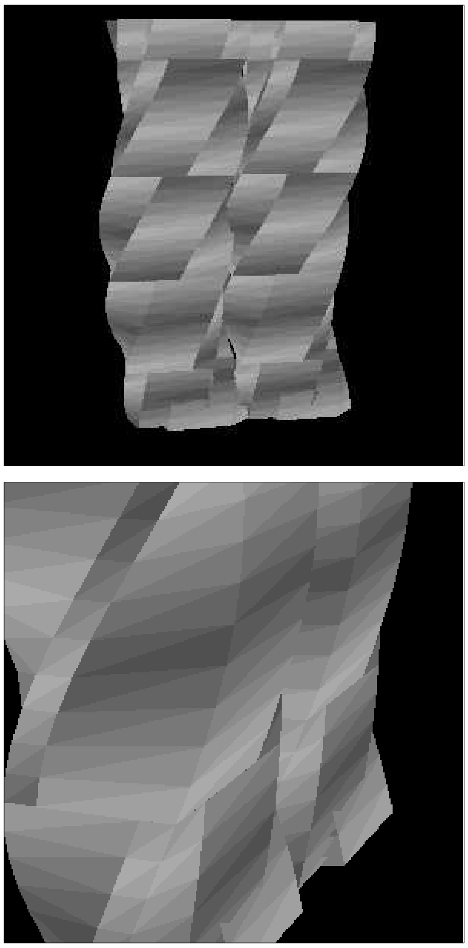

Figure 2: Two views of 2+1D C-space

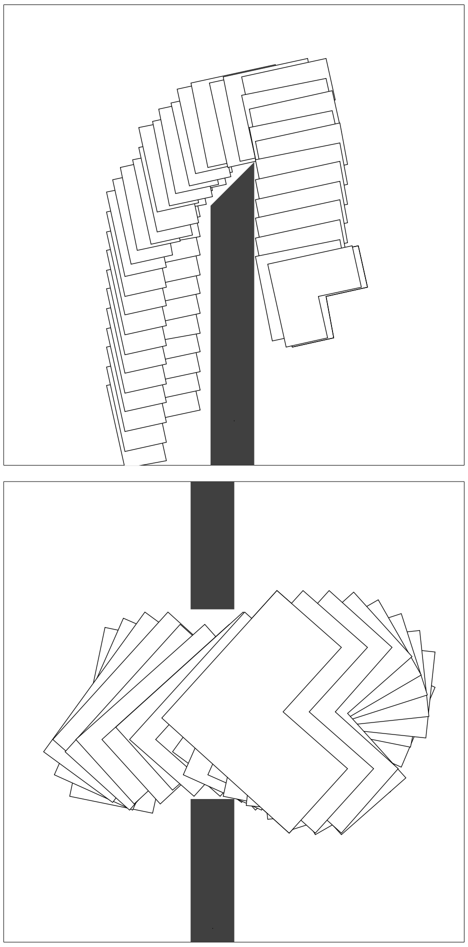

Figure 3: Paths found by FindPath

moved object, we did not succeed in deriving a single set of robust behaviors that could handle all the polygonal shapes we were interested in.

2. The second approach was to apply the principles of traditional feedback control to the problem. The idea can best be illustrated graphically. In Figure 4 we illustrate the L-shaped object's initial and final configuration at some stage along the path. A line Ol is drawn opposite the line segment connecting the positions occupied by the center of mass of the object as it moves along the path. This line intersects the line segments comprising the polygon in a number of places. The segment that contains the furthest point from O is chosen as the segment on which the pushing action will be applied. Since the direction of rotation desired is counter-clockwise, we use a rule suggested by Mason [1986] to find a point along the segment RQ and use a small incremental push dx, dy along the translation direction indicated. If no such point can be found, then we choose another edge adjacent to PQ and retry.

   This second approach was much more successful than the first in moving the objects along specified paths, although it had real trouble with producing accurate rotations. It also could not handle the entire set of ob-

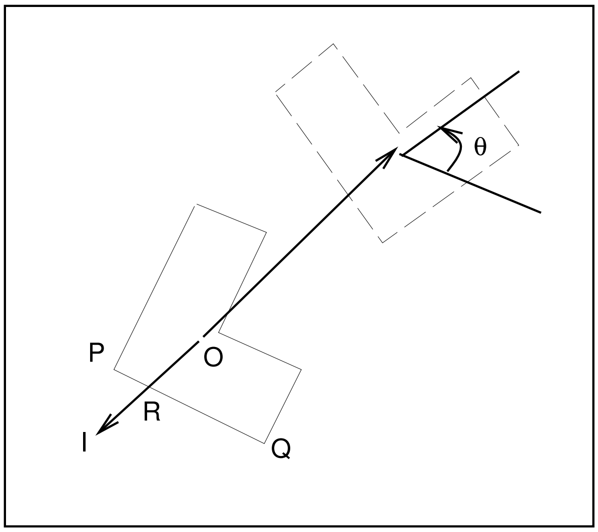

Figure 4: Geometrical Control Rule

ject shapes because the geometric rule is quite sensitive to the object shape and its configuration relative to the goal.

3. When the first two approaches failed to produce satisfactory results, we attempted to write a local controller that inverted a simulated model of the physics of the domain. The simulated model of the underlying physics assumes a randomly varying 3-point pressure distribution. Based on the theory presented in Mason [1982] we implemented a 2-d minimization procedure that takes as input an executed robot motion A_i and produces as output $ds = (\delta x, \delta y, \delta\theta)$ which is the predicted object motion if that action were to be executed. Using this, we produce a forward map from actions to changes in state by repeatedly executing this simulation procedure for various values of the action tuple.

   To implement the local controller, we invert this table. Given a desired state change we look in the table to find the action that produces the closest change we are looking for and then execute this action. We have implemented two approaches; one that minimizes euclidean distance in a normalized space and another which treats rotations differently from translations.

It should be noted that the last approach implements a feedback loop as well, but relies on a model of the physics as predicted by the simulator. However, there is no intrinsic reason why simulation should be the source for the information contained in the table. One could envision this table being built up from actual trials (see Christiansen et al. [1991] who learn such models of action empirically).

The last approach has proven to be robust in carrying out the required motions in simulations (described in more detail below). In Figure 5 we illustrate the performance of this controller in two different trials where the L-shaped object is moved from $(40, 40, 0.8)$ and $(40, 40, 1.57)$ to $(0, 0, 0)$.

We choose a particular sampling of the action space based on performance metrics. The parameters we attempt to choose are k, the number of pushing points along an edge of the object, d the length of the push and l the number of directions along which the pushing action is exerted. Values for these parameters are chosen by measuring the performance of the controller on a set of sample tasks. First a set of k equally spaced points are chosen along each edge of the pushed object. This specifies the first two values x, y of the action tuple. To derive values for the next two values, directions of pushes are sampled into l values in the range $[\hat{n} + \theta_{max}, \hat{n} - \theta_{max}]$ about this point. $\hat{n}$ denotes the normal to the edge and θ_{max} was set to 60 degrees. The length of pushes d, is at present held constant to 2 cm. By considering the number of pushes, length travelled between pushes, and the ratio of successful pushes to failures in a specified time period on a set of previously chosen pushing tasks, the algorithm chose $k = 4$, and $l = 5$ in the examples shown below. This results in the size of $\mathcal{A}$ being 120 in this case for the L-shaped object.

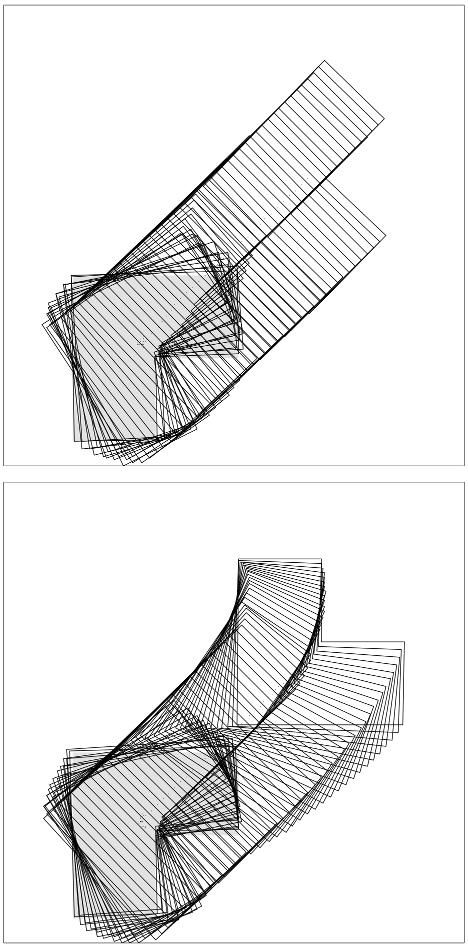

Figure 5: Two execution trails produced by local controller

6 Adding Context

If the environment had been sensed accurately, and if our model of control is perfect, and if indeed our path-planner produces collision-free paths, we would be done. A proof of completeness of our strategy synthesis approach would have two components. The first part would be to prove that our path planner is complete (always finds a path between two configuration when such a path exists) and the second part would be an asymptotic convergence proof to show that the feedback loops achieve tracking.

Unfortunately, reality is rarely this forgiving. The three sources of error often considered in the literature are:

1. *Sensor error*: The actual state s is usually assumed to be in a ball centered around the sensed state s^*. (i.e. $d(s, s^*) < S_e$ where S_e characterizes the sensing error, and d encodes the distance metric).

2. *Control error*: The action commanded is never the one that is actually executed. In our pushing domain, uncertainty in control can be modeled as an uncertainty in the position on the object's surface chosen for pushing and in the direction and length of the pushing action.

3. *Model error*: The shape of the objects in the environment is never known with perfect fidelity. Small uncertainty in an object's shape can be modeled by perturbing the coordinates of its vertices. Gross errors in shape can be modeled by introducing or delet-

ing features.

Due the sources of uncertainty, the planner and controller combination we have described above can fail to successfully execute a task in a number of ways. The most obvious failures occur because of interactions between the pushed object and other obstacles in the environment. Failures include limit cycling on segments of the path, chattering about certain points, and getting into *trap* states. In such states, the set of applicable actions that produce a noticeable effect reduces to zero.

One approach to handling this difficulty would be to attempt to model the dynamics of such interactions and include in the planner the ability to plan paths that take into account these explicit models of dynamics.

However, we will describe an approach that we have implemented that seems to work quite well under simulated models of control and sensor uncertainty. It presently cannot handle `trap` states. The approach is motivated by the observation that when collisions happen and contact occurs to change the dynamic behavior of the pushed object from what the controller expects it to do, what really matters is the locally sensed geometry of the environment. The basic idea is to generate a number of simple controllers to handle *each* of these local configurations much like the first level controller handles object motion in free-space.

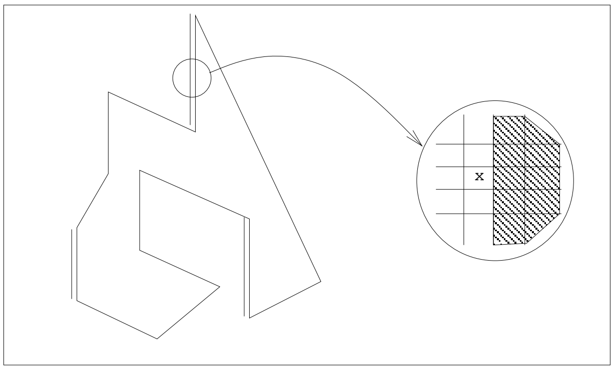

Figure 6: Illustration of Local Similarity

To illustrate the idea, consider Figure 6. Here we show a 2-D representation of a configuration space obstacle. The double lines indicate locations where the local contact geometry is similar. The expanded view indicates how if the robot is in a configuration cell marked x, its local view of the configuration space is identical if 4 or 8 neighbouring cells are considered. The reason for defining locality in configuration space should now be obvious. Since one can essentially view the pushed object as having been reduced to a point in this space, tests for locality can be much simpler even for objects of complicated shapes. If we use only orthogonal neighbours in a tesselated representation of the configuration space to define locality, then in an n-dimensional space we have 2^{2n} possible local configurations. For each one of these, our approach attempts to learn a local control rule that applies, much like the first level controller was learnt in free-space. It is this local configuration that we label *context*.

The planning, control and learning algorithm for the pushing task now looks like:

```
push-object (object, segment)
    context = sense-local-context
    controller = find-control (context)
    if ( controller = NIL )
        learn-rule (context)
    find-action (controller, object, segment)
```

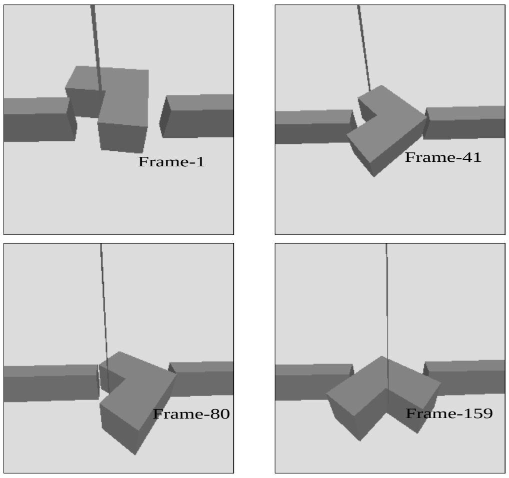

Figure 7: Four Frames from Trial Run

Such a decomposition of the task-level strategy using local context has been the key to deriving a robust strategy to handle pushing and manipulating objects of arbitrary shape (see Figure 7). This strategy has been the first one to solve all our examples we have thus far tested it on[3]. In our simulated examples, with a library of 30 part shapes on 20 tasks this approach is the only one that solves some of the complicated examples, and the only one not to exhibit significant limit cycling or chatter.

Owing to space limitations, we cannot fully describe the relationship this approach has to dynamic programming, or present a complete theoretical analysis in this paper. For these, the interested reader is referred to Narasimhan [1994]. However, it should be noted that:

1. The locally sensed configuration space allows us to implement robot relative control loops extremely easily. Reactive behaviors that attempt to follow a moving object can be expressed quite easily in this framework.
2. We do not seek to compute a single controller to handle the entire task, but settle for a set of controllers instead. We thus tread a path that is midway between approaches that seek a single sequence of actions, and approaches that attempt to derive a single map from sensed states to actions.
3. There is no inherent restriction that we only plan the path once. In fact, in domains where fast path planning can be implemented easily, we can envision the path to be the output of an any-time planner that is in continous operation.

[3] Control and angular error in our similations have been $[-10, 10]$ degrees in the direction of the push, and position sensing error has been a ball of 1 cm.

4. At present, we do not change the different controllers once they are built. However, the framework does not prevent controllers whose parameters are changeable over time, or choosing actions using different algorithms than the ones we have presently implemented.

7 Conclusion

We have presented an approach that solves the problem of pushing objects around in the plane from configuration to configuration in a robust fashion. The approach's strength lies in the combination of a global path planner along with a set of local control laws or feedback rules that are activated depending upon the local context the pushed object is in.

The global path planner can be viewed to be continually in operation, while the local controllers can be derived automatically from simulation models or actual trials. In certain domains, geometry crucially affects dynamics especially when objects interact. The success of manipulator operations hinges on taking into account such geometrical effects. Our approach relies on a decomposition which allows the planner to operate in configuration space while ignoring the dynamics and the evolution of a set of controllers to deal with local dynamics. We expect such an approach to be more successful than others that attempt to take all of the dynamics into account at planning time, and others that rely entirely on local controllers.

8 Acknowledgements

Many of the key ideas in this paper arose from discussions with Prof. Tömas Lozano-Pérez. I would also like to thank Brian Eberman and Jose Robles for their helpful suggestions, and Prof. Matt Mason at CMU for clarifying the ideas in his original paper.

9 References

1. **Baraff, D.**, *"Issues in Computing Concact Forces for Non-Penetrating Rigid Bodies"*, Algorithmica, *Vol. 10, pp. 292-352, 1993.*
2. **Brock, D. L.**, *"A Sensor Based Strategy for Automatic Robotic Grasping"*, Ph. D. Thesis, Department of Mechanical Engineering, Massachusetts Institute of Technology, *1993.*
3. **Brooks, R. A.**, *"A Robust Layered Control System for a Mobile Robot"*, IEEE Journal of Robotics and Automation, *Vol. RA-2, No. 1, pp. 14-23, March 1986.*
4. **Brooks, R. A.**, *"Intelligence Without Reason"*, Proceedings, IJCAI-91, *August, pp. 569-595, 1991.*
5. **Connell, J.**, *"A Colony Architecture for an Artificial Creature"*, MIT AI-TR-1151, Artificial Intelligence Laboratory, Massachusetts Institute of Technology, *1990.*
6. **Christiansen, A. D., Mason, M. T. and Mitchell, T. M.**, *"Learning Reliable Manipulation Strategies Without Initial Physical Models"*, Robotics and Autonomous Systems, *Vol. 8, pp. 7-18, 1991.*
7. **Erdmann, M. E.**, *"On Motion Planning with Uncertainty"*, S. M. Thesis, Department of Electrical Engineering and Computer Science, Massachusetts Institute of Technology, *1984.*
8. **Guibas, L., Ramshaw, L., and Stolfi, J.**, *"A Kinetic Framework for Computational Geometry"*, Proc. of the 24'th Annual IEEE Conference on the Foundations of Computer Science, Tucson, *pp. 100-111, 1983.*
9. **Lozano-Pérez, T.**, *"Spatial Planning: A Configuration Space Approach"*, IEEE Transactions on Computers, *Vol. C-32, No. 2, pp. 108-120, February, 1983.*
10. **Mason, M. T.**, *"Manipulator Grasping and Pushing Operations"*, Dept. of Electrical Engg. and Computer Science, Massachusetts Institute of Technology, *1982.*
11. **Mason, M. T.**, *"Mechanics and Planning of Manipulator Pushing Operations"*, International Journal of Robotics Research, *Vol. 5, No. 3, pp. 53-71, 1986.*
12. **Mason, M. T. and Wang, Y.**, *"On the inconsistency of rigid-body frictional planar mechanics"*, Proc. IEEE Conference on Robotics and Automation, *pp. 524-528, 1988.*
13. **Mataric, M.**, *"A Distributed Model for Mobile Robot Environment Learning and Navigation"*, AI-TR-1228, MIT AI Lab, Massachusetts Institute Of Technology, *1990.*
14. **Narasimhan, S.**, *"Task-Level Strategies for Robot Tasks"*, Forthcoming Ph. D. Thesis, Department of Computer Science and Electrical Engineering, Massachusetts Institute of Technology, *1994.*