Final Project Proposal - Parallelization of Robot Path Planning using MPI

Sverre Kvamme (x276063) & Simen Andresen (x276060)

February 25, 2013

Problem Formulation

For the final project in CS 140 we would like to look into the problem of finding the path in the configuration space of a 2 or 3 link robot manipulator as illustrated in Fig. 1. This involves checking for connectedness in a configuration space with arbitrary polygonal obstacles, constructing a graph over every possible path and running a shortest path algorithm.

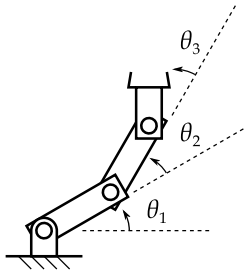

Figure 1: 3 link robot manipulator

Overview of the problem

When controlling a robot manipulator, it is important to move it in a way that is both effective and do not cause it to crash into obstacles. With both the obstacles and the robot links being arbitrary polygons this can not be done analytically and one has to use a sampling based method. This project will focus on a robot manipulator with 2 or 3 rotational joints moving in a 2 dimensional workspace. The joints are modelled as a rigid body system with 2 or 3 DOF's (degrees of freedom). Since all joints are rotational the corresponding configuration space will be a 2 or 3-Torus. With a sampled workspace giving a discrete map in the configuration space, the challenge of this problem is to construct a graph of possible edges between samples as illustrated in Fig. 2, and to find shortest paths between points in the configuration space without crashing into obstacles.

The sample based methods which are most likely to be used is either a uniform sampling of the workspace which gives equal distances between all sample points, a random sampling method or a deterministic pseudo random method called the halton sequence. With the halton sequence the samples are better distributed than the random sampling method, and gives a good method of determining upper bound that determines if two samples are connected with an edge.

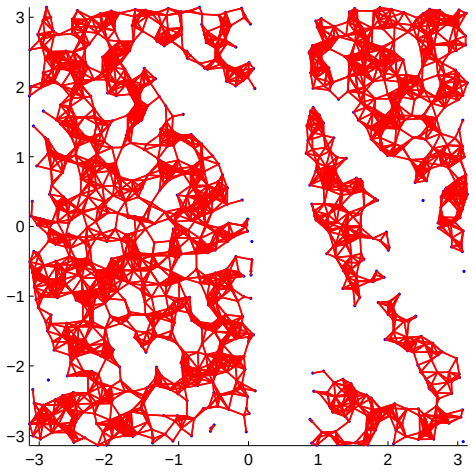

Figure 2: A sampled configuration space of a two link manipulator with edges between samples. The two axis of the figure represent the first and second angle of the links

Parallelization

When sampling the workspace of a manipulator it is very important to have a high number of samples. This is to both give effectiveness and provide safe edges that does not violate the free configuration space. In constructing a graph of edges between sample points one has to check all samples against all other giving a quadratic time complexity:

$$\frac{n(n-1)}{2} = O(n^2) \tag{1}$$

To determine if there should be edges between two samples a "Mary Go Round" strategy will be used, so that each processors determine edges based on their sample points position relative to all other sample points. Further a shortest path algorithm will be used to find the path between two arbitrary points. The program that is to be constructed will have the following input/output

Input : Number of samples, number of iterations, sampling method.

Output : A path between arbitrary points for each iteration, given as angles for each joint in the manipulator.

For getting the shortest path between points in the configuration space, a shortest path like Dijkstra's algorithm for weighted edges will be used in a parallel version.

Possible expansion of project

If the there is enough time, there are some things we would like to explore. As of now, the manipulator will take the shortest path in the configuration space. But sometimes we would like to go in a straight line in the actual workspace, i.e. we want a given point on the outermost

joint to move in a straight line, from one given point to the next. This would require some inverse kinematics to implement, but we would then be able to control more of the manipulators movement. Also, dependent on the time, we will consider how much of the program we implement in c. We already have most of a sequential code for a 2-link manipulator in Matlab. First we will only take the most time-consuming parts of the program and write it in parallel c-code, while running the other parts of the program in Matlab. The goal however is to implement the whole program in c, but simulate the result in Matlab.