Final Project Progression Report
Parallelization of Robot Path Planning using MPI

Sverre Kvamme (x276063) & Simen Andresen (x276060)
March 6, 2013

What has been done so far

So far several modules have been made as a part of solving the 3-Link Robot Manipulator problem. None of them has so far been parallelized, but has been written as sequential c-code, which has been tested against an existing matlab code with the same input. We have have so far concentrated on optimizing the sequential code with respect to run time.

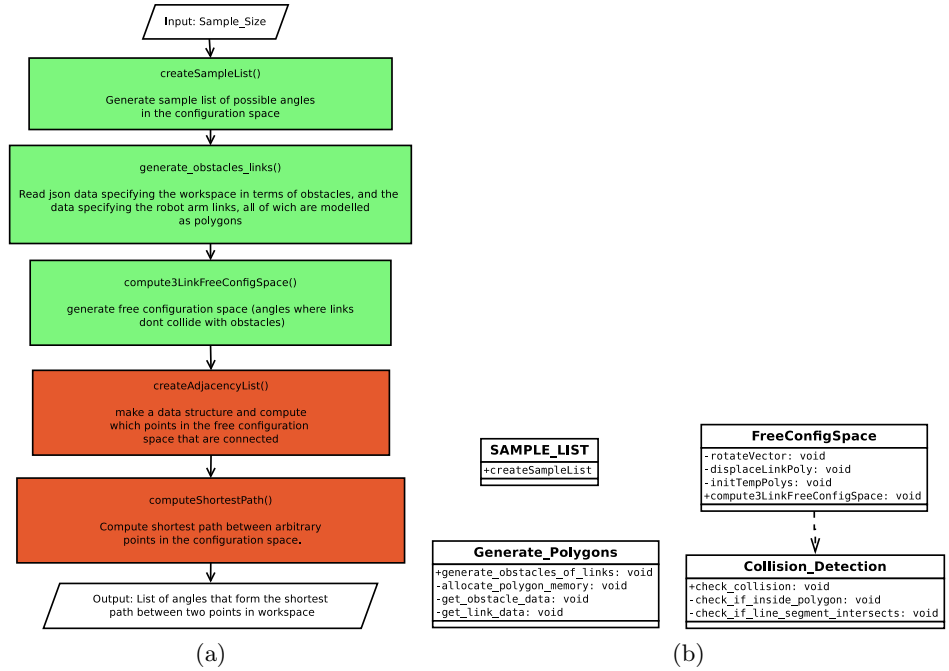

Figure 1: Program with the modules created so far marked in green. (a) and the different modules which has been implemented and their member functions and dependencies

In Fig. 1 One can see the progress on the overall program, as well as the different modules that are implemented so far.

The sampling and collision detection procedure where much less trivial than initially expected, and writing and debugging the code has been pretty time consuming. We suspect that the rest of the code will be simpler to write, at least as sequential code. Some time has also gone to making a framework for visualizing the data, both for the purpose of presentation and for debugging (and for fun). A sample of this can be seen in Fig. 2

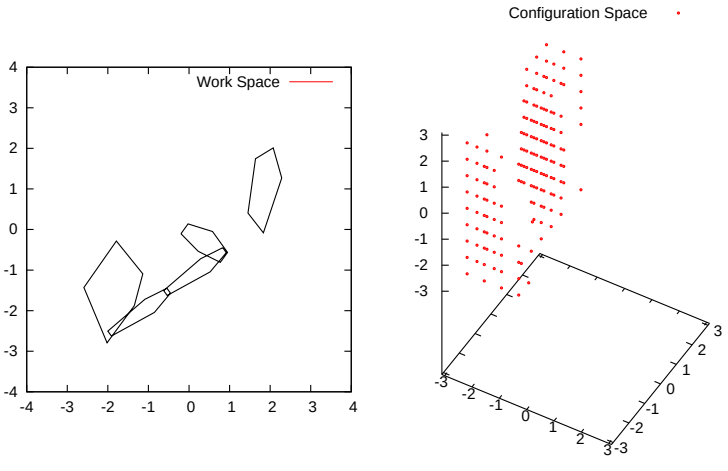

Figure 2: Sampling of configuration space by checking if manipulator collides with any of the obstacles

Changes from initial proposal

The sampling of the configuration space where initially meant to be done as a sequential procedure, but after testing the code, we found that it was very time consuming due to the fact that it has the complexity

$$kn^3 = O(n^3) \tag{1}$$

when sampling for a 3-link robot manipulator, and because the collision detection algorithm (represented as k in the equation above) is also very time consuming. We are therefore going to focus on doing the sampling procedure in parallel. The sampling procedure should be very parallelizable, due to the fact that one can divide the sample list (list of all possible angles lying on a 3 Torus) equally between processors, and each processor can then calculate their partition of the free workspace based on this.

After getting feedback from Professor Gilbert we also decided to aim at running a sequential Dijkstra's algorithm, and do the different computations in the algorithm in parallel.

Further Work

The work ahead includes computing an adjacency table which represents the graph in the problem, and then running the shortest path algorithm on the graph. The adjacency table will be computed in parallel, while the shortest path algorithm will run sequentially, with different computations running in parallel. The sampling procedure will also be extended to run in parallel. If time allows it we will implement a very realistic scenario where the workspace is changing (the obstacles are moving around) and therefor the sample procedure has to be done once per iteration. This scenario will therefore benefit from a sampling procedure running in parallel.