CS 140 - Paralell Scientific Computing

Parallel Sampling and Shortest Path in the Configuration Space of a 3 Link Robot Manipulator on a Distributed Memory Computer System using MPI

by

Sverre Kvamme & Simen Andresen

University of California, Santa Barbara

Abstract

This paper was written as a final project in the course CS 140 Parallel Scientific Computing at the University of California, Santa Barbara in March 2013. The objective of this project was to make a program for parallelizing the procedure of sampling the configuration space of a 3 Link Robot Manipulator.

When controlling robots in arbitrary environment it is necessary to sample the environment and check at which points the robot collides with obstacles, and and then construct a feasible path between points a start and goal point . This procedure is very time consuming for a robot with 3 degrees of freedom.

A solution was therefore to parallelize the procedure of sampling and path generation. This yielded a close to linear speed-up for large sampling spaces.

Contents

1 Introduction

In the field of robotics, motion planning is an important topic with many interesting challenges. One of which are the problem of deciding where a robot can move without violating constraints such as colliding with obstacles in the environment of the robot. This cannot be done analytically in closed form, and sampling of the environment must be done to check where the robot can move in order to comply with these constraints. When it is decided where the robot can move, one can use a shortest path algorithms to decide how to move from a desired start point to a desired goal point. This procedure can be very computational intensive, and using parallel computation would therefore increase the running time, allowing for more samples, and thus higher accuracy and efficiency of the robot motion.

This paper addresses this problem for a specific scenario of a 3 link robot manipulator shown in Fig. 1 where the procedure of sampling, collision detection and shortest path are written in the C programming language using Message Passing Interface (MPI) for parallelizing.

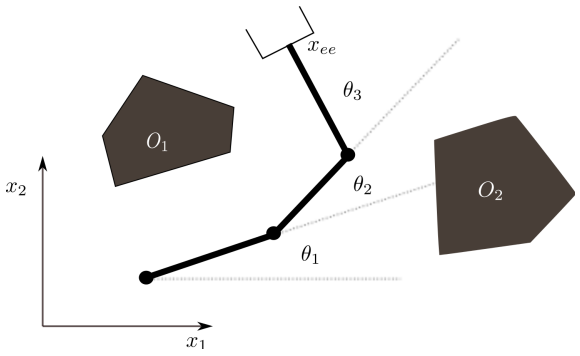

Figure 1: Illustration of a 3 link robot manipulator moving in a two dimensional space

2 Notation and Preliminaries

Further some specifications and notes on notation can be useful.

- The **Workspace W** is the set of all $\mathbf{x} = \begin{bmatrix} x_1 & x_2 \end{bmatrix}^{\mathbf{T}}$ coordinates in the physical environment of the robot, and is defined as

$$W \in \mathbb{R}^2 \tag{1}$$

- The **Configuration Space C** is the set of all coordinates $\boldsymbol{\theta} = \begin{bmatrix} \theta_1 & \theta_2 & \theta_3 \end{bmatrix}^T$ and belongs to the three dimensional space

$$C \in \mathbb{S}^1 \times \mathbb{S}^1 \times \mathbb{S}^1 = \mathbb{T}^3 \tag{2}$$

  Where $\mathbb{T}^3$ denotes that the configuration space lies on a 3-torus. Since a 3 torus is difficult to visualize, the configuration space will be visualized as a cube as illustrated in Fig. 2. It should further be noted that since $\theta_i \in \mathbb{S} = [-\pi, \pi)$ and $-\pi = \pi$ each side of the cube is the same as it's opposite side.

- The **Free Configuration Space $\mathbf{C}_f$** is the subset of points in C which does not cause collision between any link of the robot and an obstacle in the workspace.
- The links and obstacles O_1 and O_2 are modelled as convex polygons, where the property of convexity facilitates the collision detection.

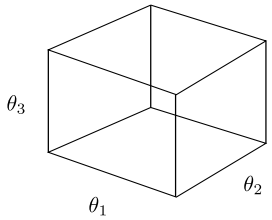

Figure 2: Illustration of the configuration space of the 3-link robot visualized as a cube.

3 Overview of the Program

The program is divided into 7 modules each with it's own purpose. The modules and their respective member functions are illustrated in Fig. 3. Having a look at the program running on one processor, one can divide the program into 4 steps as illustrated in Fig. 4 and the program will be explained in this top-to-bottom order.

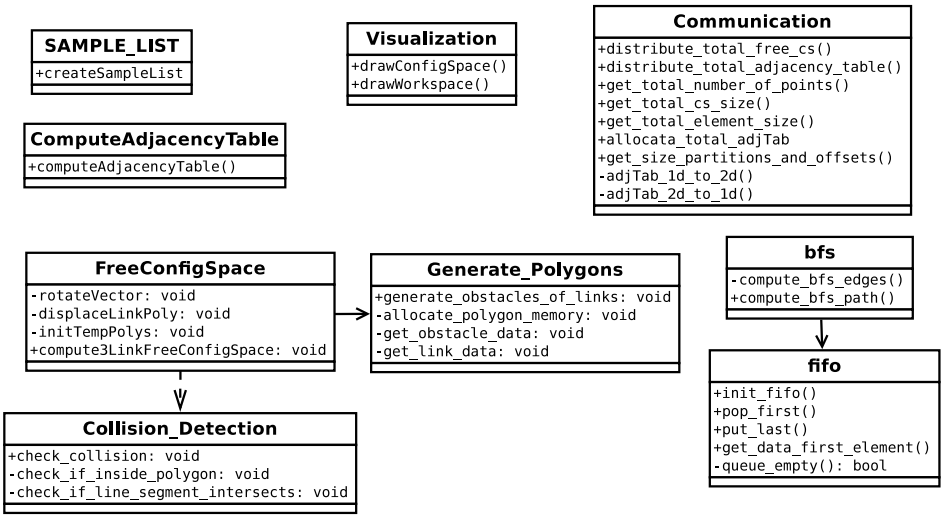

Figure 3: Diagram showing the different modules and dependencies

4 The Program

In this section each of the different main procedures will be explained.

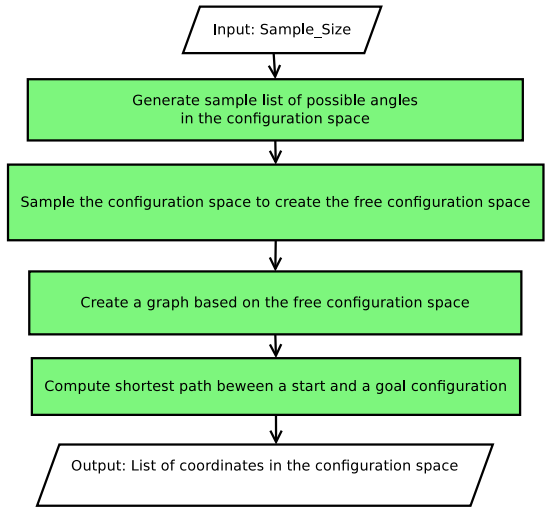

Figure 4: Float chart showing the sequential run of the program

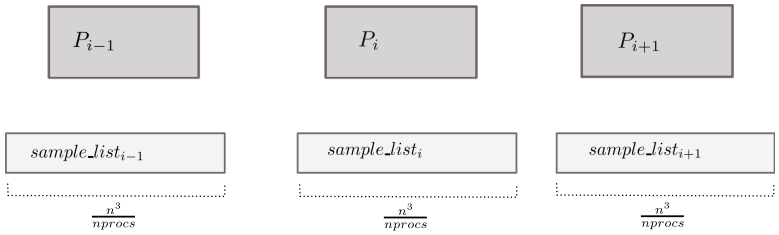

Figure 5

Listing 1:

```
int a=b;
```