Optimizing Actuator Network Positions

Robert L. Read *

Founder, Public Invention, an educational non-profit.

April 28, 2017

1 Introduction

The Gluss Project builds robots that are networks of linear actuators connected with concentric joints. A fundamental problem of contronlling such robots is to move one of the joints, or nodes, to a given position. More generally, we would like to leave some nodes (the feet) in position which we move another set of nodes (other feet, or graspers) to some positions that we specify. In general, this may require a motion of every actuator in the network.

Design of efficient gaits and motion depends on this abiity. The ability to crawl over obstacles depends on this ability.

Although the current robot is a tetrahelix, a structure isomorphic to the Boerdijk–Coxeter helix, we would eventually like to build robots and machines with general geometries, including non-tetrahedral geometries, and will develop the algorithm with that in mind.

This problem is dependent on the physical nature of the actuator: it can change length between and minimum and maximum length. This problem is silimar to optimization problems such as solved by linear programming. However, at a minimum the constraints are quadratic.

The mininum length and maximum length are defined by the Cartesian distance formula, which expressed as a length contains a square root, but we may always work with the square of this formula, leaving quadratic formualae. I'm pretty sure is is a subcase of Conic Quadratic optimization: http://docs.mosek.com/modeling-cookbook/cqo.html because we are dealing with Euclidean norms.

Note we are dealing with positive definite matrices.

*read.robert@gmail.com

2 Formulation

We now define the problem ACTNETOPT.

The input to our problem can be formalized as:

- An input dimension d (probably 2 or 3.) Nodes positions are elemenst of $\mathbb{R}^d$
- A model of a net of n nodes in the form of a graph.
- A mapping from from node index i to minimum length $Y(i)$ and maximum length $Z(i)$.
- A set of goal points g_j associated with node j chosen from $\mathbb{R}^d$.
- A linear weighting $w(j)$ of goal points interpreted as the cost of not placing the node j at the goal point g_j proportional to the square of that distance.
- A set of static nodes S which may not be moved by the algorithm.

We now attempt to formulate the problem as a QCQP. The quadratically constrained quadratic programming problem can be formulated:

$$\begin{aligned} \text{minimize: } & \frac{1}{2}x^T P_0 x + q_0^T x \\ \text{subject to: } & \frac{1}{2}x^T P_i x + q_i^T x + r_i \leq 0 \end{aligned}$$

In ACTNETOPT case we have no equality constraints.

In order to create this in matrix form, we create from it:

- A model of our robot represented by a symmetric matrix $M^{n\times n}$, where n is the number of joints in the robot, and a $M_{i,j} = 1$ if the nodes i and j are connected by an acutator, and is zero if not. Elements of M are real-valued.
- Similar matrices $Y^{n\times n}$ and $Z^{n\times n}$ representing respectively the minimum and quadrance (square of the distance) for each actuator i, j. If actuator i, j does not exist in the robot, then $Y_{i,j}$ and $Z_{i,j}$ are undefined. (Note that $Y_{i,j} = 0$ is an interesting case. Furthermore we are particularly interested in the case then all Y and M values are equal where they are defined.) Elements of Y and Z are real-valued.
- A set of goal points g_j associated with node j chosen from $\mathbb{R}^d$.
- A goal matrix $P_0^{n\times n}$ which is positive definite and represents a potentially weighted sum of the square of the Euclidean norm, or quadrance, of the position of nodes in our x_i from their respective goal postions g_i.

The output of the algorithm is a vertical vector X which satisifies all contraints and minimizes the objective function f.

To the author it was not obvious how to construct the matrices in question, so it is perhaps worth stating this explicity. Let us consider a single upper bound constraint, $z_{i,j}$. z_i is a scale, x_i is a 2- or 3-vector.

$$|\mathbf{x_i} - \mathbf{x_j}| \leq z_{i,j}$$
$$(x_{i_x} - x_{j_x})^2 + (x_{i_y} - x_{j_y})^2 + (x_{i_z} - x_{j_z})^2 \leq z_{i,j}$$
$$x_{i_x}^2 + -2x_{i_x}x_{j_x} + x_{j_x}^2 + x_{i_y}^2 + -2x_{i_y}x_{j_y} + x_{j_y}^2 + x_{i_z}^2 + -2x_{i_z}x_{j_z} + x_{j_z}^2 \leq z_{i,j}$$

Which can be represented in the matrix P_i, assume $i = 0$, $j = 2$, the 3-vectors are unpacked linearly.

$$\begin{pmatrix} 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & -2 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix}$$

This matrix has 9 entries representing the nine coefficients. In fact it is positive definite (proof?).

So: $x^T P_i x + r_i \leq 0$ represents a distance constraint when r_i is the square of the mininum distances for actuator i.

The goal matrix P_0 can be constructed similarly, multiplying each set of 9 entries by a $w(j)$ to weight that node.

Since there is one P_i for each actuator and each such matrix has only nine entries, this entire approach is very sparse. In the fact the constraint matrices are so specialized that we can expect an algorithm specific to this problem to do well. Since most of the quadratic constraint programming packages are commercial, there is a strong incentive to develop a specific algorithm.

3 Algorithm

Although this problem is a quadratic constraint problem, is is highly specialized, and such solvers are not freely available. Because our robots are at present models (the current robot has 24 actuators), it is reasonable to assume this nut can be cracked with a small hammer.

4 References