Soft Body Locomotion: Supplementary Document

Jie Tan Greg Turk C. Karen Liu

Georgia Institute of Technology

Abstract

This supplementary document describes implementation details of a QPCC solver for contact modeling. A simple example is provided to help the readers test their implementation.

1 QPCC for Contact

The QPCC problem for contact modeling is:

$$\min_{\dot{\mathbf{p}}^{n+1},\mathbf{u},\mathbf{f}_\perp,\mathbf{f}_\parallel,\lambda} G(\dot{\mathbf{p}}^{n+1},\mathbf{u}) \tag{1}$$

subject to

$$\mathbf{u}_{lb} \le \mathbf{u} \le \mathbf{u}_{ub}$$

$$\widetilde{\mathbf{M}}\dot{\mathbf{p}}^{n+1} = \mathbf{f}_c + \mathbf{A}\mathbf{u} + \widetilde{\mathbf{f}}^n \tag{2}$$

$$\mathbf{0} \le \begin{bmatrix} \mathbf{f}_\perp \\ \mathbf{f}_\parallel \\ \lambda \end{bmatrix} \perp \begin{bmatrix} \mathbf{N}^T\dot{\mathbf{p}}^{n+1} \\ \mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda \\ \mu\mathbf{f}_\perp - \mathbf{E}^T\mathbf{f}_\parallel \end{bmatrix} \ge \mathbf{0} \tag{3}$$

where $G(\dot{\mathbf{p}}^{n+1},\mathbf{u})$ is a convex quadratic objective function of next velocities $\dot{\mathbf{p}}^{n+1}$ and control variables $\mathbf{u}$, which are bounded by $\mathbf{u}_{lb}$ and $\mathbf{u}_{ub}$.

Equation 2 is the discretized dynamic equation (Equation 8 in the paper), where $\widetilde{\mathbf{M}}$ is the mass matrix with terms from implicit integrator, $\mathbf{f}_c$ and $\mathbf{A}\mathbf{u}$ are the contact force and control force (scaled by the time step) respectively, and $\widetilde{\mathbf{f}}^n$ accounts for all other terms in the dynamic equation.

Equation 3 is the LCP formulation to regulate contact velocity and contact force, $\mathbf{f}_c = \mathbf{N}f_\perp + \mathbf{D}\mathbf{f}_\parallel$, where $\mathbf{N}$ is the unit normal vector, $\mathbf{D}$ is a set of tangential directions at the contact point, and $f_\perp$ and $\mathbf{f}_\parallel$ are the magnitudes of normal and tangent forces. μ is the friction coefficient and λ is an auxiliary variable whose physical meaning is related to the tangent velocity of a sliding contact. This form is slightly more general than the QPCC formulation (Equation 11) in the paper.

2 Implementation of QPCC Solver for Contact

Algorithm 1 summarizes the implementation of our QPCC solver. The solver takes the QPCC (Equation 1-3) as input and outputs the minimizer $\mathbf{x}^*$ and minimum objective function value f^*. Our iterative QPCC solver starts with an initial guess of the contact situation, which is a set of linear constraints that are compatible with the complementarity conditions. Function *GenerateInitialGuess* in Line 8 generates such an initial guess. We usually choose static contact

Algorithm 1: Pseudo-code of the QPCC solver.

```
1  (x*, f*) = Solve(QPCC)
2  begin
3      ithIter = 0;
4      f* ← ∞;
5      x* ← null;
6      priority_queue ← [];
7      visitedQP_set ← {};
8      CCSpec ← GenerateInitialGuess();
9      QP ← GenerateQP(QPCC, CCSpec);
10     (QP.minimizer, QP.fval) ← QP.Solve();
11     priority_queue.Enqueue(QP);
12     while visitedQP_set.Size() < maxNumVisitedQP and
       priority_queue.Empty() = False do
13         QP ← priority_queue.Dequeue();
14         visitedQP_set.Add(QP);
15         if QP.fval < f* then
16             f* ← QP.fval;
17             x* ← QP.minimizer;
18         childQP_list ← GenerateChildQP(QPCC, QP, CCSpec);
19         foreach childQP in childQP_list do
20             if visitedQP_set.Find(childQP) then
21                 continue;
22             (childQP.minimizer, childQP.fval) ←
               childQP.Solve();
23             priority_queue.Enqueue(childQP);
24         ithIter ← ithIter + 1;
25     return (x*, f*);
```

situation as the initial guess:

$$\mathbf{0} \le \mathbf{f}_\perp, \qquad \mathbf{N}^T\dot{\mathbf{p}}^{n+1} = \mathbf{0}$$

$$\mathbf{0} \le \mathbf{f}_\parallel, \qquad \mathbf{D}^T\dot{\mathbf{p}}^{n+1} + \mathbf{E}\lambda = \mathbf{0}$$

$$\mathbf{0} = \lambda, \qquad \mu\mathbf{f}_\perp - \mathbf{E}^T\mathbf{f}_\parallel \ge \mathbf{0}$$

Occasionally, this initial guess is inconsistent with the dynamic constraints (Equation 2), resulting in an infeasible problem. In such a case, we solve the Mixed LCP problem (Equation 2 and 3), assuming $\mathbf{u} = \mathbf{0}$, to obtain a feasible initial guess. The output of *GenerateInitialGuess* is a boolean array *CCSpec* that specifies a set of linear constraints from the complementarity conditions. If the ith entry of *CCSpec* is True, we choose the linear constraints $Cond_1 = 0$ and $Cond_2 \ge 0$ for the ith pair of complementarity conditions $0 \le Cond_1 \perp Cond_2 \ge 0$. If it is False, we choose $Cond_1 \ge 0$ and $Cond_2 = 0$.

Function *GenerateQP* in Line 9 generates a QP by replacing the complementarity conditions of QPCC with the linear constraints specified in *CCSpec*. In Line 10, we solve the initial QP and record its minimizer and minimal function value. We add this QP to an priority queue, which sorts the QP's by their function values in an

increasing order. Line 12 starts the iteration, which terminates until the number of explored QP's exceeds a threshold or the priority queue is empty.

In each iteration, we retrieve the QP at the head of the queue (Line 13), add it to the set of explored QP's and update f^* and $\mathbf{x}^*$ if necessary (Line 14-17). Function *GenerateChildQP* generates a list of new QP's by pivoting complementarity conditions (Line 18), which we will discuss in the next paragraph. For each new QP in the list, if it has been already explored, we discard it (Line 20-21). Otherwise, we solve the new QP and add it to the queue (Line 22-23). When the algorithm terminates, we output the best minimizer and function value that we found during the iterations (Line 25).

We summarize the detail of *GenerateChildQP* in Algorithm 2. The input of *GenerateChildQP* includes QPCC, current QP and its constraint specification for complementarity conditions: *CCSpec*. In Line 4, we extract the solution for each contact from the minimizer of the QP. Function *ExtractFromMinimizer* selects the correct components associated with a specific contact point from the minimizer. In Line 6, we identify the index of the normal force-velocity condition pair for the ith contact, where *IdInCC* returns the index of a specific pair among all complementarity conditions. Line 7 identifies the case of contact establishment and performs the corresponding pivoting (Line 8). Line 11 checks whether the contact is about to break and pivots the constraint accordingly, which enables the contact breakage (Line 12). Line 15-16 identify the indices of friction cone and friction direction conditions associated with the ith contact. If the static friction force has reached the boundary of friction cone, we switch the contact situation from static to sliding (Line 17-18) and estimate the sliding friction direction using the static friction direction (Line 19-23). We apply the opposite pivoting (from sliding to static) when the sliding velocity reaches zero (Line 26-29). A new QP is generated under the new contact situation and added into the list of new QP's (Line 9-10, 13-14, 24-25 and 30-31) whenever pivoting happens. After processing through all the contact points, the function returns the new QP list.

Algorithm 2: Generate a list of new QP's based on the minimizer of the current QP

```
1  QP_list = GenerateChildQP(QPCC, QP, CCSpec)
2  begin
3      foreach i in Contacts.Size() do
4          (ṗ, f_⊥^i, f_‖^i, λ^i) ← ExtractFromMinimizer(QP.minimizer);
5          newCCSpec ← CCSpec;
6          id ← IdInCC(0 ≤ f_⊥^i ⊥ (N^i)^T ṗ ≥ 0);
7          if CCSpec[id] = True and (N^i)^T ṗ=0 then
8              newCCSpec[id] ← False;
9              newQP ← GenerateQP(QPCC, newCCSpec);
10             QP_list.Add(newQP);
11         else if CCSpec[id] = False and f_⊥^i =0 then
12             newCCSpec[id] ← True;
13             newQP ← GenerateQP(QPCC, newCCSpec);
14             QP_list.Add(newQP);
15         id ← IdInCC(0 ≤ λ_⊥^i ⊥ μf_⊥^i − (E^i)^T f_‖^i ≥ 0);
16         id_list ← IdInCC(0 ≤ f_‖^i ⊥ (D^i)^T ṗ + E^i λ^i ≥ 0);
17         if CCSpec[id] = True and μf_⊥^i − (E^i)^T f_‖^i = 0 then
18             newCCSpec[id] = False;
19             foreach id1 in id_list do
20                 newCCSpec[id1] ← True;
21             f ← D^i f_‖^i;
22             j ← argmax_k f^T (D^i.Col(k));
23             newCCSpec[j] ← False;
24             newQP ← GenerateQP(QPCC, newCCSpec);
25             QP_list.Add(newQP);
26         else if CCSpec[id] = False and λ^i = 0 then
27             newCCSpec[id] ← True;
28             foreach id1 in id_list do
29                 newCCSpec[id1] ← False;
30             newQP ← GenerateQP(QPCC, newCCSpec);
31             QP_list.Add(newQP);
32     return QP_list;
```

3 Test Case

It is nontrivial to test and debug an implementation of a QPCC solver. We propose the following test case problem: A particle with mass m lies on the ground and it wants to jump up. The particle can only control a jumping force $f\mathbf{N}$, where $\mathbf{N} = (0, 1, 0)$ is the up direction. The magnitude of the jumping force is bounded by $0 \leq f \leq f_{ub}$. What is the optimal jumping force if the goal is to jump as high as possible? The answer is trivially using maximal force possible. However, the solution is not necessarily f_{ub} under some contact configuration. Our hope is that the QPCC solver can find the optimal contact configuration such that the optimal solution reaches f_{ub}.

One formulation of the above problem is:

$$\min_{\dot{\mathbf{p}}, f, f_\perp, \mathbf{f}_\parallel, \lambda} -f^2$$

subject to

$$0 \leq f \leq f_{ub}$$
$$m\dot{\mathbf{p}} = \Delta t(m\mathbf{g} + \mathbf{N}f_\perp + \mathbf{D}\mathbf{f}_\parallel + \mathbf{N}f)$$
$$\mathbf{0} \leq \begin{bmatrix} f_\perp \\ \mathbf{f}_\parallel \\ \lambda \end{bmatrix} \perp \begin{bmatrix} \mathbf{N}^T\dot{\mathbf{p}} \\ \mathbf{D}^T\dot{\mathbf{p}} + \mathbf{E}\lambda \\ \mu f_\perp - \mathbf{E}^T\mathbf{f}_\parallel \end{bmatrix} \geq \mathbf{0}$$

where Δt is the time step used in the simulation and $\mathbf{g}$ is gravity.

It is easy to verify that the minimizer is

$$(\dot{\mathbf{p}}, f, f_\perp, \mathbf{f}_\parallel, \lambda) = (\frac{\Delta t}{m}(m\mathbf{g} + \mathbf{N}f_{ub}), f_{ub}, 0, \mathbf{0}, 0)$$

and the optimal function value is $-f_{ub}^2$. A correctly implemented QPCC solver based on Algorithm 1 and 2 should converge to this optimal solution in two iterations. In the first iteration, an initial QP using static contact situation is generated and solved. The solution should be

$$(\dot{\mathbf{p}}, f, f_\perp, \mathbf{f}_\parallel, \lambda) = (0, |m\mathbf{g}|, 0, \mathbf{0}, 0)$$

The function *GenerateChildQP* will pivot the constraint to switch from static contact to contact breakage and generate a new QP. In the second iteration, the solution of this new QP is exactly the optimal solution of the QPCC problem.

Once the implementation passes this simple test case, more challenging cases should be tested. For example, control a single rigid body, an articulated rigid-body system, or a soft body with the presence of contact.