A sparse model predictive control formulation for walking motion generation

Dimitar Dimitrov[1], Alexander Sherikov[1], Pierre-Brice Wieber[2]

[1] Örebro University, Sweden

[2] INRIA, Grenoble, France

September 10, 2011

Scenario

- humanoid robot walks on a $\underbrace{\text{flat surface}}_{\text{assumption}}$
- the system could be subject to external disturbances
- higher level planner generates reference footsteps

Objective

follow reference footsteps while preserving the "stability" of the system

Required

a scheme for online trajectory following and stabilization

Trajectory following + stabilization $\triangleq$ walking motion generation

How to approach the problem (in under 2 ms)

- using predefining motion primitives - not possible in the presence of disturbances
- making local decisions considering the full dynamical morel - not reliable
- "look-ahead" schemes - increasingly popular but computationally demanding. In particular, using full system dynamics - not feasible.

One possible "solution"

- use approximate dynamical model (preferably linear)
- compensate the approximation by applying a preview type of controller with (possibly) fast sampling rate

We use

- **model**: linearized 3D inverted pendulum - surprisingly accurate approximation (under certain assumptions)
- **preview controller**: Linear Quadratic Regulator (LQR) with explicit constraints $\triangleq$ Linear Model Predictive Control (LMPC)
- **stability criterion**: ZMP $\in$ support polygon

Explicit constraints - address the stabilization sub-task

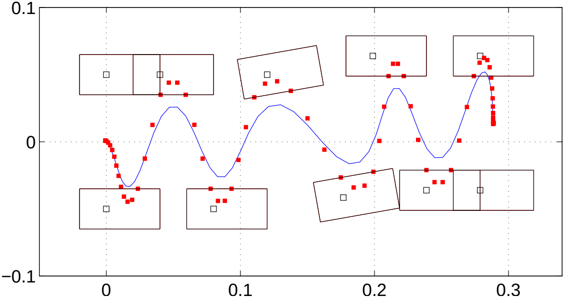


Figure: A typical result (fixed feet). Red squares - ZMP, blue line - CoM. Double support constraints are not displayed.

The paper deals with efficient implementation

Linear dynamical system

$$\boldsymbol{x}_{k+1} = \mathbb{A}\boldsymbol{x}_k + \mathbb{B}\boldsymbol{u}_k$$

$\boldsymbol{x}_0$ is a known initial state

$k = 0, \dots, N-1$

Quadratic objective function (to minimize)

$$J(\boldsymbol{v}_x, \boldsymbol{v}_u) = \boldsymbol{v}_x^T \boldsymbol{H}_x \boldsymbol{v}_x + \boldsymbol{v}_u^T \boldsymbol{H}_u \boldsymbol{v}_u$$

number of variables: $N_x + N_u$

$$\boldsymbol{v}_x = (\boldsymbol{x}_1, \dots, \boldsymbol{x}_N), \quad \boldsymbol{v}_u = (\boldsymbol{u}_0, \dots, \boldsymbol{u}_{N-1})$$

The “standard approach”

Reformulate problem using minimal number of variables N_u. Or in other words, eliminate equality constraints due to the dynamics of the system.

$$\boldsymbol{v}_x = \underbrace{\begin{bmatrix} \mathbb{A} \\ \mathbb{A}^2 \\ \vdots \\ \mathbb{A}^N \end{bmatrix}}_{\boldsymbol{w}} \boldsymbol{x}_0 + \underbrace{\begin{bmatrix} \mathbb{B} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbb{A}\mathbb{B} & \mathbb{B} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbb{A}^{N-1}\mathbb{B} & \mathbb{A}^{N-2}\mathbb{B} & \dots & \mathbb{B} \end{bmatrix}}_{\boldsymbol{W}} \boldsymbol{v}_u$$

$$J(\boldsymbol{v}_u) = (\boldsymbol{w}\boldsymbol{x}_0 + \boldsymbol{W}\boldsymbol{v}_u)^T \boldsymbol{H}_x (\boldsymbol{w}\boldsymbol{x}_0 + \boldsymbol{W}\boldsymbol{v}_u) + \boldsymbol{v}_u^T \boldsymbol{H}_u \boldsymbol{v}_u$$

Some drawbacks of eliminating the equality constraints

$$\underset{\boldsymbol{v}_u}{\text{minimize}} \quad J(\boldsymbol{v}_u) = (\boldsymbol{w}\boldsymbol{x}_0 + \boldsymbol{W}\boldsymbol{v}_u)^T \boldsymbol{H}_x (\boldsymbol{w}\boldsymbol{x}_0 + \boldsymbol{W}\boldsymbol{v}_u) + \boldsymbol{v}_u^T \boldsymbol{H}_u \boldsymbol{v}_u$$

$$= \boldsymbol{v}_u^T \underbrace{\left(\boldsymbol{W}^T \boldsymbol{H}_x \boldsymbol{W} + \boldsymbol{H}_u\right)}_{\boldsymbol{H}} \boldsymbol{v}_u + \dots$$

1. the new Hessian matrix $\boldsymbol{H}$ is in general **dense** - the structure of the problem is lost
2. forming the product $\boldsymbol{W}^T \boldsymbol{H}_x \boldsymbol{W}$ is expensive and usually has to be performed offline
3. the computational cost per iteration is $O(N^3)$ (for *interior-point* methods) and $O(N^2)$ (for *active-set* methods)

In the context of our application the matrices $\boldsymbol{H}_x$ and $\boldsymbol{W}$ contain information about

- discretization of the preview window
- height of the CoM

Point 2 above implies that both should be constant.

Sparse formulation

As opposed to *condensing* the problem, one can solve directly

$$\begin{aligned}
\underset{\boldsymbol{v}_x,\,\boldsymbol{v}_u}{\text{minimize}} \quad & \begin{bmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_u \end{bmatrix}^T \begin{bmatrix} \boldsymbol{H}_x & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{H}_u \end{bmatrix} \begin{bmatrix} \boldsymbol{v}_x \\ \boldsymbol{v}_u \end{bmatrix} \\
\text{subject to} \quad & \boldsymbol{x}_{k+1} = \mathbb{A}\boldsymbol{x}_k + \mathbb{B}\boldsymbol{u}_k, \quad k = 0, \dots, N-1 \\
& \boldsymbol{x}_0 \text{ is a known initial state.}
\end{aligned}$$

This is a larger but more structured quadratic program (QP)

- solution can be obtained at a cost of $O(N)$ per iteration ($e.g.$, by using Riccati recursion)
- there is no need to pre-compute the objective function offline
- CoM height and discretization sampling can be altered online (at practically no additional computational cost)

How about inequality constraints?

We perform a change of variable that leads to a simplified formulation

Standard approach

- **control input**: jerk of CoM
- **output**: position of ZMP

jerk of CoM → system dynamics → position of ZMP

⇒ The system dynamics appears in the constraints for the ZMP

We use the ZMP directly as a decision variable

$$\begin{aligned} \underset{\text{ZMP}}{\text{minimize}} \quad & \textbf{usual stuff} \\ \text{subject to} \quad & \text{ZMP} \in \text{support polygon} \leftarrow \text{pure geometry} \end{aligned}$$

In this way we can derive a formulation with

- simple bounds
- diagonal Hessian matrix

Numerical results (active set method)

Preview window 1.5 s

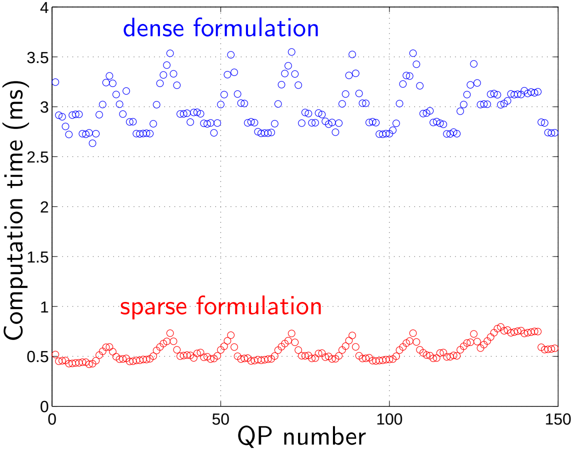
- N = 75
- T = 20 ms

Dense formulation (off-the-shelf solver QL)

- # variables: 150
- # eq. constraints: 0
- # simple bounds: 150

Sparse formulation (custom-made solver)

- # variables: 600
- # eq. constraints: 450
- # simple bounds: 150



C++ implementation available for download at

https://github.com/asherikov/smpc_solver