A framework for manipulation and locomotion with realtime footstep replanning
Dang Duong, Florent Lamiraux, Jean-Paul Laumond

To cite this version:

HAL Id: hal-00697564
https://hal.archives-ouvertes.fr/hal-00697564
Submitted on 15 May 2012

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire HAL, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.
Abstract—This paper focuses on realization of tasks with locomotion on humanoid robots. Locomotion and whole body movement are resolved as one unique problem. The same planner and controller are used for both stages of the movement. Final posture and footprint placements are found by resolving an optimization problem on the robot augmented by its footprints. Footstep replanning is done in real-time to correct perception and execution errors. The framework is demonstrated with the HRP-2 robot in a number of different scenarios.

Index Terms—locomotion, footsteps, replanning, real-time, visual servoing

I. INTRODUCTION

LOCOMOTION and manipulation are among the most exciting topics in humanoid robotics research. Numerous works have been carried out in the past years in walking and running generation [1]–[7], notably with the introduction of the zero momentum point (ZMP). On manipulation, task-based methods have been developed since the eighties of the last century for industrial robots and robotic arms, [8], [9]. These methods have been extended to humanoids in recent years as more and more robots have been made available for research [10]–[12].

While both locomotion and manipulation generate on their own great interests for research and can be treated separately as is often the case, it is intriguing how much one can extend the capability of robots when combining these two problems in the same context. With locomotion coupled with manipulation, the robot is no longer constrained to manipulate only objects in its workspace. In fact, the workspace is extended with locomotion that just becomes another task. In addition to creating new capability and new task, there is a real need to build the bridge between locomotion and manipulation. There are many cases where the coupling is so important that one cannot separate both problems. A humanoid robot might need to make steps before manipulating some objects. A dynamically demanding manipulation scenario might require that the robot end up at some exact configuration at the end of the locomotion stage. For instance, to manipulate an object at ground level, constraints have to be put on feet placement and affect the way locomotion should be conducted.

Resolving locomotion and a complex upper body manipulation is, however, not a simple task in practice. Typically, a computationally expensive planning stage is involved at first. The resulting plan may be expressed either in configuration space, which can be executed directly on the robot, or in terms of tasks to be given to the controller. Yet, no sensing system is perfect, the information obtained at the planning stage is usually different from reality. In addition, execution errors, due for instance to sliding while walking make open loop execution unrealistic. As a consequence, correction has to be done online. This is a challenging task since a time-consuming replanning can prevent the robot from being reactive. Online generation of footsteps have been studied previously by several research groups [13]–[15] but usually in the context of a decoupled locomotion problem.

In this paper, a framework is proposed to treat both manipulation and locomotion with online replanning. Sources of errors such as perception system, execution error, moving objective are taken into account online by the motion planner, which in turn, produces in real time up-to-date instructions to the controller in the form of task space trajectories for the feet (steps) and for the upper-body (manipulation).

Approach and contribution

The motion planner in this paper extends the work of Kanoun et al. [16], [17], where robot motion is resolved in the form of an optimization problem. The humanoid robot is augmented by its footprints and tasks are expressed on this augmented robot. Modifications have been made to the initial method in order to make it run on-line. First, instead of resolving the problem in the configuration space of the robot, only an intermediate state of the resolution is performed and fed to the controller: namely final posture and foot placements during walking.

The main contribution of this paper is the integration of the method described in [16] into a control framework that
achieves real time replanning during walking. The entire framework is then validated with a number of different scenarios on the dynamic simulator OpenHRP and validated on the real robot with the scenario described in Section IV-C (Figure 1).

II. IMPLEMENTATION

A. Architecture overview

Figure 2 depicts how different components interact with each other. At first, a complete plan is made and fed to the system. This is done only once hence the dotted arrow. In the execution loop, images from the robot cameras are transmitted to the perception module. This module analyses the images to detect and localize objects of interest in the environment with respect to the robot. The visual servo uses 3D position of tracked object to replan in real time upcoming footprints and upper body tasks (hand task, posture task, etc.). Footprints are passed to a preview controller which also computes trajectories of critical operation points of the lower body, which is responsible for walking or keep the robot balanced while standing, namely, trajectories of left foot, right foot and the center of mass. The controller combines these tasks all together within a prioritized stack and computes the appropriate control signal that is sent to the robot. Since the controller makes no distinction between walking and object manipulation, it only executes tasks, there is no "gap" between the walking and manipulation stages.

B. Footstep and posture planning

The core of the planner is the resolution of a hierarchical system of inequality and equality tasks [16], [17]. The special feature of this approach is the possibility of having inequality tasks in higher priority than equality task. This is extremely helpful in humanoid robots where a lot of contraints have to be expressed as inequality tasks.

1) Prioritized linear and inequality systems: Considering a system of linear equalities and inequalities:

\[
Ax = b
\]

\[
Cx \leq d
\]

for which the solution can be expressed as an optimization problem

\[
\min_{x \in \Omega} \frac{1}{2} ||Ax - b||^2 + \frac{1}{2} ||w||^2
\]

subject to

\[
Cx - w \leq d
\]

In an prioritized linear system, one can resolve 3 iteratively at each level as follows:

\[
S_0 = \mathbb{R}^n
\]

\[
S_{k+1} = \arg \min_{x \in \Omega} \min_{w \in \mathbb{R}^n} \frac{1}{2} ||Ax - b||^2 + \frac{1}{2} ||w||^2
\]

subject to

\[
S_{k}x - w \leq d_k
\]

In robotics, a task is typically represented by its Jacobian \( J \). The classic example is a position task in operational space with \( T(q) = 0 \) is the goal and \( T(q) = c \) the current error. The solution \( \dot{q} \) to achieve exponential convergence \( c = C_0 \exp(-\lambda t) \) is written as

\[
J\dot{q} = -\lambda c \quad J = \frac{\partial T}{\partial q}(q)
\]

Other tasks can be written as inequalities \( T(q) \leq 0 \) such as collision and self-collision avoidance, joint limits, etc. The resolution of a prioritized set of task can be computed by:

Find \( \dot{q}^* \in S_k \):

\[
S_0 = \mathbb{R}^n
\]

\[
S_i = \arg \left\{ \min_{\dot{q} \in S_{i-1}} \frac{1}{2} ||J_i\dot{q} - e_i||^2 \right\} \text{ for equality tasks}
\]

\[
S_i = \arg \left\{ \min_{w \in S_{i-1}} \frac{1}{2} ||w||^2 \text{ s.t. } J_i\dot{q} - e_i \leq w \right\} \text{ for inequality tasks}
\]

2) Application to footstep planning: As explained earlier, the determination of footsteps and final posture defining the motion executed on the robot is achieved in one problem by using the previous resolution of equality and inequality system. The main idea of this approach is to add a virtual joint (Figure 3) with three degree of freedoms for each step. Let us consider a robot of \( n \) degrees of freedom executing \( k \) steps, the resulting virtual kinematic chain adds \( 3k \) degrees of freedom to the real robot to form a \( n + 3k \) d.o.f. kinematic chain. Constraints on how far the robot can physically step or turn become joint limits for theses new joints. The constraints that the robot should not step one foot on one another becomes auto-collision avoidance. On this new robot with this set of constraints in addition, a set of appropriate tasks can then be applied, namely:
inequality constraints, in order, joint limits, projection of
the center of mass, self-collision avoidance of robot, self-
collision avoidance for the virtual manipulator, position
and orientation of the supporting foot
• object manipulation task. e.g. grasp, reach for an object
with 2 hands, etc.
• parallel task for the upper body during walking,
• gaze task (keep objects of interest in the vision field).
Figure 5 depicts how the upper body task "attracts" the
augmented robot hence initiate footsteps. The last \( n \) degrees
of freedoms in the result represent the final posture, and
the final position of the virtual kinematic chain represents
footsteps. The complete motion in configuration space can
then be found by passing the footsteps to a preview controller
which output trajectories for left foot, right foot and the center
of mass. These trajectories in turn, in addition with the upper
body task at the final step can be given as equality tasks to
the prioritized stack. The resulting motion is therefore never
computed before-hand. It is instead the result of the control
architecture.

Determine the number of steps: In the previous section,
it was assumed that the number of steps \( k \) was known before
hand. However, the target detected by the vision system can be
found in a wide range of distance. Therefore, it is difficult to
guess in advance how many steps the robot should make. One
important point in this section and also the first modification
made on this paper to the original method is the fact this
parameter \( k \) can be found automatically by the optimizer. The
algorithm is summarized as follows:

Figure 5. depicts the exploration of the virtual kinematic
chain in space.

III. ONLINE FOOTSTEP REPLANNING

One big challenge in our context is the use of stereo-vision
to detect and localize objects. Localization errors grow dra-
matically with the distance between the object and the sensor.
Usually, this distance is maximal in the initial configuration
of the robot, when motion planning is performed.

Algorithm 1 Footsteps planning with variable number of steps
Require: initial position.
Require: manipulation task.
Require: obstacle positions.
Ensure: footprints and final posture
Initialize solver with 0 virtual link.
repeat
    Solve the optimization problem.
    if Reach local minimum then
        Add new virtual link
    end if
    Check manipulation task error
until goal reached

The resolution to a complete motion in joint space described
in previous section typically takes up to one minute for a
long motion. Yet, to be reactive, replanning should finish
at least once every stepping interval of the robot (typically
under a second). It is clear that replanning the motion up to
configuration space motion is not feasible in practice.

A. Footsteps and final posture replanning

As explained earlier intermediate results, i.e. final posture
and footsteps, of the planning stage are sufficient for the
controller. This is where replanning becomes feasible since
the computation of footsteps and posture is typically from \( 3 \) to
\( 10s \), a replanning could be well below \( 1s \) and hence guarantee
reactivity. The augmented robot then starts at previous state
and update its tasks according to new sensory information
about landmarks. Algorithm 2 describes the stages of planning,
the results are shown in Figure 6. The solver converges
more quickly than planning footsteps and posture from initial
position since the current configuration is already closed to
the goal configuration. Table I shows replanning time for
a grasping task with goal as a ball on the ground at a
distance around 2 meters. The modification is taken randomly
in arbitrary directions.
The replanning process for the whole-body and footsteps depends greatly on configuration and it is not guarantied a computing time less than stepping time (0.8s). Without modification, small corrections (e.g. due to drifting) can be dealt with. Otherwise, an execution strategy must be applied at the planning-control bridge to deal with large errors, such as the case of stereo vision system.

**Stop and go:** One obvious approach to address the realtime issue is to stop the robot or let it step in place if the planner takes too long to respond.

**Interactive optimisation:** As shown in Section II-B, the complexe robot tends towards the goal during the optimitisation process. At early stages, i.e. when the robot starts, having the full solution with all footsteps and final pose is not necessary. In fact, even a full solution is found, it is highly likely that this solution will change during walking motion to compensate sensory errors. The planner can keep up with the controller by only replan, at each correction, to an intermediate state. Namely, line 14 in Algorithm 2 will be modified to: "replan to new solution or timeout", where the timeout chosen is one stepping period.

While stop and go strategy guarantees optimised trajectory at each replanning stage, it produces unnatural movements on the robot. Using iterative approach, the robot does not have to stop while correcting its paths towards the new goal. However, this method relies on the assumption that the intermediate stage found at each timeout is safe for the robot.

As it turns out, there exists a third strategy which consists in reducing the dimension of the optimisation problem in replanning stage by blocking some or all joints belonging to the upper body. At each replanning step, optimised foot placements and posture are found without compromising the time constraint. Without having to interrupt the optimisation process at each timeout as the second approach, this method is also simpler to implement on the robot. This third strategy is chosen for experiment and is detailed in the following section.

### B. Realtime footstep replanning

While footsteps have to be changed to correct execution drift or perception error, the posture does not. In fact, unless new obstacles are found near the goal the previous posture stays valid. For example, if the robot is to approach and open a door, even the position of the door on its reference has change, the previous posture still guarantees dynamic equilibium. This scenario covers most cases in experiment. The most part of the last $n$ degrees of freedom of the the augmented robot can be 'frozen' while other joints are free to move. Suppose $l$ degrees of freedom are locked. This translates into reducing the dimension of the Jacobians, hence the complexity of the problem.

Table II shows replanning time in a grasping scenario similar to the one described in previous section with the posture of the standing robot "frozen". This is the other extreme case compared to Table I where every joint in the standing robot is free to move. In the first replanning scheme, the robot can barely correct small errors, so the robot has to make many small corrections at a time, hence a large amounts of supplement steps have to be added to the locomotion stage. In the second replanning schema however, the replan time is much less than time necessary for the robot to make a step. This guarantees a realtime replanning running behind the scene and updating robot footprints continuously.

**Extension to footstep replanning of an arbitrary initial path:** In cases where all upper-body joints are blocked in the replanning phase, this method can be extended to a variety of motion. In fact, the inputs for the planner is just initial footsteps and postures, which can be given by a different planner than the one described in section II-B, for example, an RRT planner. The robot can therefore deals with more complexe cases, typically scenarios where the optimisation method fails due to local minimas, for example, a long wall is placed between the robot and goal.

#### Algorithm 2 Footprint replanning

**Require:** current plan.

**Ensure:** new plan

1: **loop**
2: replan needed ← false
3: check position of target in camera
4: check current step number
5: if Goal position changes then
6: Update gaze and hand task
7: replan needed ← true
8: end if
9: if current step number changes then
10: Update virtual manipulator in solver
11: replan needed ← true
12: end if
13: if replan needed then
14: Replan to new solution
15: end if
16: end loop

<table>
<thead>
<tr>
<th>Goal modification (m)</th>
<th>Max (ms)</th>
<th>Min (ms)</th>
<th>Average (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>2615</td>
<td>251</td>
<td>595.1</td>
</tr>
<tr>
<td>0.02</td>
<td>2159</td>
<td>275</td>
<td>673.2</td>
</tr>
<tr>
<td>0.05</td>
<td>2343</td>
<td>488</td>
<td>920.7</td>
</tr>
<tr>
<td>0.1</td>
<td>2498</td>
<td>593</td>
<td>1299.8</td>
</tr>
<tr>
<td>0.2</td>
<td>4166</td>
<td>608</td>
<td>1977.0</td>
</tr>
<tr>
<td>0.5</td>
<td>7123</td>
<td>610</td>
<td>3685.3</td>
</tr>
</tbody>
</table>

**TABLE I**

CPU time for replanning of final posture and footsteps (milliseconds) for a locomotion-grasp task (Figure 1)

<table>
<thead>
<tr>
<th>Goal modification (m)</th>
<th>Max (ms)</th>
<th>Min (ms)</th>
<th>Average (ms)</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.01</td>
<td>16</td>
<td>8</td>
<td>10.7</td>
</tr>
<tr>
<td>0.02</td>
<td>38</td>
<td>7</td>
<td>11.4</td>
</tr>
<tr>
<td>0.05</td>
<td>12</td>
<td>9</td>
<td>11.0</td>
</tr>
<tr>
<td>0.1</td>
<td>48</td>
<td>9</td>
<td>15.5</td>
</tr>
<tr>
<td>0.2</td>
<td>23</td>
<td>11</td>
<td>20.0</td>
</tr>
<tr>
<td>0.5</td>
<td>117</td>
<td>15</td>
<td>33.6</td>
</tr>
</tbody>
</table>

**TABLE II**

CPU time for replanning footsteps (milliseconds) for a locomotion-grasp task for a locomotion-grasp task (Figure 1)
C. Control

With the plan (final posture and footsteps) being updated in realtime, it is up to the controller to execute the plan. In this framework the "stack-of-tasks" [18]–[20] is used. The principle of task-based control is similar to the formulation of optimisation problem in Section II-B.

In this framework the control law $\dot{q}_i$ is written as following:

$$\dot{q}_i = -\lambda J_i^+ e_i$$  \hspace{1cm} (9)

$J_i^+$ is the pseudo-inverse of the Jacobian $J_i$ [9]. $e_i$ is the difference between desired feature $s_i^*$ (i.e. a position in operational space, a posture, etc.) and its current value $s_i$:

$$e_i = s_i - s_i^*$$  \hspace{1cm} (10)

The control law on a set of tasks is written [21]:

$$\dot{q}_i = \dot{q}_{i-1} + \lambda_i \tilde{J}_i^+ (e_i - J_i \dot{q}_{i-1})$$  \hspace{1cm} (11)

when $\tilde{J}_i$ is the projection of $J_i$ in the null space of the augmented Jacobian

$$J_i^A = [J_1, J_2, \ldots, J_{i-1}]^T$$  \hspace{1cm} (12)

$$\dot{\tilde{J}}_i = J_i P_{i-1}$$  \hspace{1cm} (13)

The remedy here is activating the posture task first, then using a simple heuristic to activate hand task only after sometime at an appropriate time. In practice, placing a threshold on the hand task error and activating the hand task accordingly provides good results.

IV. Experiments

Described here are some scenarios with the humanoid robot HRP-2 using the proposed method.

A. Reach and open a door

In this scenario, (Figure 9) the robot has to walk towards a door, reaching for the handle and pull to open the door by walking backwards.

B. Catching a flying balloon

The robot must follow and catch a falling balloon. (Figure 10). No assumption is made about the dynamics of the balloon. The robot simply try to place its hands at certain height at the right position to catch the ball. In simulation, a user alters the wind condition to change the trajectory of the balloon hand changing robot planned motion online.

C. Grasping a moving ball on the ground at distance

This example illustrates probably the best the connection between locomotion and manipulation since precise conditions have to be met for the robot footsteps to make the grasping...
dynamically stable. Similar to the catching scenario, the position of the ball changes during walking and the robot must adapt to this change (Figure 1).

V. CONCLUSION REMARKS AND FUTURE WORKS

A generic method for footstep and posture planning and real time replanning has been presented in this paper. Numerous scenarios have demonstrated the applicability of this framework. All demonstrated motion have been tested dynamically with the simulator OpenHRP. On the real robot, the goal detection and tracking using using CAMShift [23] algorithm acts as the perception module. Experiments have been conducted on the robot with HRP-2 following and reaching a ball on the ground, with the goal eventually moved during experiment.

Expansion can be made to this scheme of footsteps replanning in more dynamically intensive cases, e.g. catching a volleyball instead of a balloon. In this cases, more precise model for the ZMP will have to be investigated to make fast steps possible. Modifications will also need to be done in the feasibility constraints of the virtual kinematic chain.

REFERENCES


