Planning support contact-points for humanoid robots and experiments on HRP-2

Adrien Escande
IBISC CNRS, France
Email: adrien.escande@ensmp.fr

Abderrahmane Kheddar
JRL CNRS, Japan
Email: kheddar@ieee.org

Sylvain Miossec
JRL AIST, Japan
Email: sylvain.miossec@aist.go.jp

***Abstract*— This paper deals with the motion planning of a poly-articulated robotic system for which support contacts are allowed to occur between any part of the body and any part of the environment. Starting with a description of the environment and of a target, it computes a sequence of postures that allow our system to reach its target. We describe a very generic architecture of this planner, highly modular, as well as a first implementation of it. We then present our results, both simulations and real experiments, for a simple grasping task using the HRP-2 humanoid robot.**

I. INTRODUCTION

Nowadays, humanoids robots are able to walk on horizontal plane, slightly sloped, sliding, or uneven terrains Some manage to climb stairs or even run (like Honda's Asimo). However, in most of the cases, these robots move using only their feet, what reduces the amount of possible motions. On the contrary, we humans often use other parts of our body either to help a biped motion, for example by increasing the stability, or to perform motions that wouldn't have been possible with a usual upright biped posture. Some works address the issue of increasing stability using a part of the robot: for example Harada et al. by computing a modified ZMP allowed an HRP-2 humanoid robot to climb a high stair while grasping a handrail [1]. But the use of this part is not the result of a planner, it is imposed at the beginning of the motion.

In a more general way, mobile motion planning deals mostly with finding a free collision path to reach a given spot [2]. Various algorithms have been developed for the 2D case, some of which have been successfully extended to the 3D poly-articulated case [3]. Although planning for task and navigation in the 3D space for poly-articulated robots is not trivial, the animation of human-like figures is even more challenging: the outcome of the planner needs not only to find a plausible solution, but also a natural-looking motion. For digital actors, Yamane et al. [4] proposed an animator that combines path planning techniques with domain knowledge data-driven. Their result generates nice animations of virtual avatar manipulating various objects. The planner is capable to find a motion for the manipulated object such that the obtained poses satisfy several geometric, kinematics, and pose constraints. However, all these actual solutions, as sophisticated as they can be, have been developed to generate trajectories free from any collision with the environment.

In this work, we on the contrary address the motion planning of poly-articulated robots having contacts between any part of their body and the environment. This means that we allow and even explicitly seek for collisions if they can help to enhance the motion. This problem requires a specific approach. Indeed, it is different from the general path planning where a path is to be find free from collisions. It is also different from the traditional planning of contact motion on the C-obstacles. Even if this last requires exactness comparing to the first one, only rigid bodies have been considered [5]. We also considered the problem as a particular case of multi-robots cooperative manipulations planning. But this idea appears to be not plausible because: (i) the problem of multi-robot cooperative manipulations is continuous whereas our problem is discreet, and (ii) most of work does not deal with the possibility to take support on the C-space, which requires precision in the generated path.

Since we have interdependency between a trajectory planning and the planning of adding or breaking contacts, we tried to see if we can use the work of Simeon et al. presented in [6]: they proposed a manipulation planning allowing grasping and re-grasping operation where the grasps are chosen on a continuous set. This approach could be used if we consider that we need to plan contacts and breaks of contact for a single part of the robot, other engaged contacts being considered as fixed. However each time we would like to change the part we are interested in, we would have a new instance of the problem since the fixed contacts would change, and, by doing so, the configuration-space of the robot changes. Moreover, we miss a part of the planner that will chose which part of the robot we consider to have/break a contact, and how and when we chose another part.

Our problem has already been explored by Hauser et al. in [7]. The overall idea is the same, namely contact-before-motion planning: planning is made in the sets of contacts space, and then from the chosen sequence of contacts, the motion is computed. However we do not search the sets of contacts space in the same way: in [7], a graph of set of contacts is built and searched. Two adjacent nodes are two sets of contacts whose difference is a single contact and for which a numeric solver can find a posture of the robot, that satisfy the contact constraints of both sets. All possible contacts (that is a point from the environment and a point of the robot) are given at the beginning. We, on the contrary, incrementally build a tree, according to a potential-like function, whose new nodes are generated from the previous one. Moreover, our posture

solver enables us to ask for rather natural-looking posture.

This paper is organized according to the following plan: we first present the general structure of our planner as well as the underlying notions and problems. Then we show a first implementation of each module. This is followed by a simulation and a manipulation with the humanoid robot HRP-2 [8], whose results are discussed.

II. Overview of the contact support planner

We consider a poly-articulated robot performing a desired task in a given environment. In this environment, we predefined some obstacles to be potential supports. In the most general way, contacts are defined by a set of constraints. For example, we ask for an arbitrary point of the surface of the robot to correspond to an arbitrary point of the surface of an obstacle among the predefine ones. We can also ask a point of the robot to be part of a surface of the environment. This last would be a sliding contact.

Definition 1 (Contact switch): is said to define either the creation of a new contact or a break of an existing contact. This corresponds to a new configuration of the robot.

Definition 2 (New state): In our planner, a new state corresponds to the creation or the breaking of a single contact.

Our planner relies on 2 main layers: a tree builder/explorer and a posture generator. The first one acts in the space of sets of contacts, while the second one attempt to find a collision-free stable posture for a given set of contacts, and optimize this posture according to some criterions. The result of this planner is a list of postures that are used as *keys* for trajectories generation. In this part we will describe the first layer and its main functions.

A. The tree builder/explorer

This is the upper part of our planner that gives its overall behaviour. It is inspired by the Best First Planning algorithm (BFP) presented in [2]. BFP algorithm is a potential-field based algorithm efficiently working on low-dimensional discretized configuration spaces (dimension less than 6). Roughly speaking, it generates and evaluates, at each iteration, the neighbour configurations of the best (according to the potential field) non-visited configuration it built so far. This way, it escapes local minima by filling them.

In a similar way, our algorithm creates neighbours to a given set of contacts. Neighbours are other sets of contacts that differ from the actual one of exactly one contact and are reachable from the actual one. The pseudo-code of the algorithm is given in the boxed text.

In this pseudo-code, function *insert*(leaf, list) puts *leaf* in *list*, according to its evaluation, *first*(list) removes the first element of *list* and returns it, and *empty*(list) returns true if *list* is empty, false otherwise.

The proposed planner architecture is completely generic: the data in the node can be anything as long as we provide the correct functions *generateSons*, *trajectoryExists*, *allowsToReachGoal* and *evaluate*. The function *backtrackPath*(node) just returns the succession of nodes from the initial one to the final one by going from a node to his father until the initial one is reached and then reversing the obtained path. *GenerateSons* is the most complex function.

Algorithm: Contact-support planner: Pseudo-code

Data: Robot models, Environment, Target
Result: List of postures
-leavesList: *a sorted list, according to the evaluation of a leaf.*
-n and n': *are nodes*
-newLeaves: *are a list of leaves*

leavesList $\leftarrow \oslash$
insert(init, leavesList)

ContactPointPlanner(node init)
begin
  while *(!empty (leavesList))* **do**
    $n \leftarrow$ *first*(leavesList)
    newLeaves $\leftarrow$ *generateSons*(n)
    for *each node* $n' \in$ *newLeaves* **do**
      if *trajectoryExists*(n') **then**
        if *allowsToReachGoal*(n') **then**
          return (*backtrackPath*(n'))
        end
        evaluate(n')
        insert(n', leavesList)
      end
    end
  end
  return failed
end

From a previously feasible set of contacts, *GenerateSons* should return a set of new states that are feasible for the robot, regarding the joint limits, a stability criterion, and the collisions (self collisions [9] and collisions with obstacles). A set of contacts is feasible if we can generate a posture that verifies the above constraints along with the contact constraints.

B. Target function

The target state can be defined in various (but classical) ways. In this planner, it is implemented by defining one of several points on given selected bodies to reach a desired position, eventually with a predefined orientation. The target can also be defined with a desired posture in the joint space and placing the overall robot in 3D space. Nevertheless, not all joints, position or orientation need to be predefined. We prior check that there is no conflicting constraint which prohibits the realization of the target state. We also check that the desired target corresponds to a controllable equilibrium.

The *AllowsToReachGoal* function calls the posture generator that tries to find a posture satisfying all the constraints of the current state (contacts, stability, etc.) as well as the target constraints. If a posture is found, then the goal can be reached from the current state. The target function is independent

from the rest of the planner; it can be rewritten and designed differently. This is called at each new state to check whether the goal is reached or not.

C. Metric

After defining the desired target, it is necessary to have a metric (in the sense of a mathematic measure) which will allow estimating how far we are from the target. This metric must guarantee to the robot to converge toward the task by a succession of intermediate postures corresponding to contact switches. This metric would ideally be a criterion to engender a motion toward the goal. Subsequently, a generated motion that decreases an associated distance would be a plausible one.

But, we are using the metric only as a sufficient indicator, not a necessary one. That means that in some cases the distance may increase before decreasing again. For example if we take the distance between the goal and the current gravity centre of the robot, the distance will increase while the robot is bypassing an obstacle between the goal and the starting position. The *evaluate*(node) function is the counterpart of the potential field in BFP: it gives a mark to node, according to the chosen metric, that allows inserting it correctly in the sorted list.

D. Trajectory

The *TrajectoryExists* function ensures that there is a path between the two consecutive computed postures of two consecutives nodes. It can be made by using existing classical planners. General motion planning between two generic configurations for a high-dimensional robot as a humanoid robot is still a difficult problem. However, some planers exist and since we consider two consecutive postures, the path between them is likely to be rather simple.

E. Neighbours

In our case the neighbours of a given set of contacts are the sets of contacts that differ from it of exactly one contact point. Generating a new surface-surface contact can be seen as choosing a point in a subspace of a 5-dimensional space: 2 coordinates for a point of the surface of the robot, 2 for a point of the environment and 1 parameter to specify a relative orientation. Thus, neighbours have to be found in a 5-dimensional space, as it would be for BFP while planning the path of a 5-DoFs robot.

III. Implementation

We present in this section the implementations of our modules. We underline that we took very simple implementations in order to validate the structure of our algorithm.

A. Posture generator

This is the lower part of our planner. Its input is a set of contacts and its output a configuration of the robot that is stable, collision-free, and that verifies the constraints of the contacts. This part was inspired by the work done within the HuMAnS toolbox provided by INRIA. We used this toolbox to generate with Maple the C-code we needed to make all geometrical computations (mostly analytical transformation matrices and position of the CoM). We thus perform optimized geometrical computations for a robot, but we need to have a different C-code for each robot.

Using this generated code, the computation of the configuration relies on the CFSQP library, which is the C version of the FSQP algorithm. FSQP is a powerful numerical solver optimizing smooth objective functions under general smooth constraints. In our algorithm, we translate all contacts into constraints as well as the stability criterion or the collision checks. For example, a fixed face-to-face contact consists in three constraints: coincidences of a point and two orthogonal vectors of robot with a point and two orthogonal vectors of the obstacle. Points are obviously taken on the surfaces of both objects. A set of contacts becomes thus a set of constraints, what allows considering very generic contacts as long as they can be translated into smooth constraints.

Once all the constraints are generated, we call the FSQP routines. If we do not set an objective function, and we don't need it, the solver just tries to find a solution that verify all constraints. Nevertheless, objective functions may lead to more natural-looking postures, or postures that ensure easier further movements. For example we first tried to minimize the distance between the joint angles and the middle of their limits, to avoid singularities. We then chose to optimize the distance to the average between this middle and 0, having more natural-looking postures while still avoiding joint limits. Within the HuMAnS toolbox, which also uses FSQP to generate posture, the solver tries to minimize the torques that fight gravity. Experiments showed that the result looks even more natural, but the computation time is also much slower.

B. Functions implementation

The *GenerateSons* function lets thus appear 4 sub-modules: *generatePosture*, *generateCandidateForContact*, *stabilityCriterion* and *checkCollisions*.

- *GeneratePosture* is the function encapsulating the posture generator.
- *stabilityCriterion*: the simplest and conservative one is to verify the existence of a torque solution capable of balancing external forces while being within the interval of torque limitation and while the external contact forces remain within their friction cone.
  But as long as the contacts are on horizontal plans, we used the CoM stability criterion (projection of CoM onto an horizontal plane lies within the convex hull of the projection of the contact surfaces onto the same plane). We are aware that this criterion is only a necessary and sufficient condition of (quasi-) static stability if all the contact surfaces are on the same horizontal plane [10], but in the case of horizontal contacts, it is a sufficient condition, what fits a preliminary implementation. This second criterion is much more simpler and gives less constraints, what makes the posture generation a bit faster.

- The *generateCandidateForContact* function proposes new contacts by choosing: (i) a body (restricted in a first implementation to feet, hands and upper part of the legs) from which it takes a predefined point, (ii) a point on the contact surfaces within a subset of the surfaces for which the contact point is likely to be reachable, and (iii) an orientation around the normal of the contact surfaces, when contact are made by the hands or feet.
  GenerateCandidateForContact tries to remove existing contacts too, when the contact has not been generated during the last step. For each proposed new configuration, we then check its feasibility with *generatePosture*: we consider that the body of the new or the removed contact is arbitrarily closed to its desired or former position and orientation, but not really in contact. For example, if we want the hand to be put on the table, we compute the posture just before the hand touches the table. Thus, we ensure that if a posture is found, the new contact is in the intersection of the contact space and the stability space of the former configuration.
- *checkCollisions* gives the posture generator the constraints of collisions. However, we didn't implement the self-collision checking for our first scenarios since the objective function helps to obtain self-collision free configuration as long as the workspace is not too complex (narrow spaces for example).

For the implementation of the *evaluate*(node) function, we chose to use the distance between the centre of mass (CoM) of the robot and the goal, or a point representing this goal. While it may not decrease at each step of a successful path, this function does give an overall indication of the movement: we normally need to bring the CoM of the robot toward the goal to be able to reach it.

The *TrajectoryExists* function was not implemented for the time being: we assume that there always exist a path between a configuration and its father: as we choose our new contact points in the stability space of a configuration, this is likely to be true. We can nevertheless find examples in which the two configurations are in C-free but there is no C-free path between them. These cases are non common. So far, we just generate the trajectory after the planning, using the different computed configurations derived from the output as key frames.

IV. Simulation with the HRP-2 humanoid

A. A grasping task instance

The skeleton of the algorithm is implemented with the modules composing the general architecture. As an instance of poly-articulated robot we used the 3D model of the humanoid robot developed by Kawada Industries. Details on technical specifications for this humanoid can been obtained in [8].

The mission consists for the humanoid HRP-2 to bring a can put on a table. Only a part of the mission is experimented. Indeed, the robot is put near the table with a given initial posture. Then it has to reach the can, grasp it and return to its initial posture.

B. Simulation and experimental results

The snapshots in the figure 1 illustrates the outcome of our planner. Here the table is considered altogether as an obstacle and a potential support. That is to say, in our algorithm, the table is considered as a support for the current robot's part that is candidate to create a contact and also for the parts which are already in contact with it. It is then considered as obstacle for the remaining robot parts.

Starting for the configuration illustrated in snapshot 1 of figure 1, *AllowsToReachGoal* function fails in finding a posture that is able to reach the goal under specified constraints in terms of stability and non collision. Then the planner will seek for a support contact that allows the humanoid getting closer to the target. As a first candidate, the right gripper is chosen. The sub-algorithm which delimits the contact support area for each body is actually simple but this part is actually being refined in the continuation of this work. The potential contact points are randomly projected. This is also to be reconsidered. Nevertheless, when the point is chosen by the *generateCandidateForContact* function the planner executes all the steps allowing to generate this new configuration and also the motion which realizes this new configuration under the predefined stability criteria, see the snapshot number 4 in figure 1.

At the second level of the tree, the *AllowsToReachGoal* function still returns false. This means that the optimization program failed again in finding a plausible posture which allows the HRP-2 to reach the target. In this case, the upper part of the left leg is tried as candidate.

The snapshot 4 of figure 1 shows the posture in which the leg enters in contact with the table and a new contact is made. At the next step, the *AllowsToReachGoal* function fails. This might be due to an over constrained configuration (since we want to find a posture which holds existing contacts). Since the algorithm allows also breaking contacts (we recall that one of the conditions is that the contact to be removed should not be the one taken in the last step), the left hand contact is removed with a final posture which allowed the robot to reach the target while keeping its equilibrium. This is illustrated in snapshot 5, 6, 7 and 8. However light inter-penetration may occur between the leg and the table. This is explained by the fact that we constraint only one contact point on the leg and the table, and by the unavoidable numerical errors that occurs. The can is then grasped and the robot played the reverse path to back to its the initial posture ! with the can being grasped.

When the simulation has been confirmed with several similar trials, we ported the obtained trajectories from the planner on the actual robot HRP-2. The speed of the robot has been reduced for security reasons. Motions are generated for each contact configuration without smoothing: i.e. the postures end with nearly zero speed. The stabilizer of the HRP-2 is disabled because it has not been conceived to handle multi-contact configurations.

The figure 2 illustrates snapshots from the real experiments. The HRP-2 executed the given trajectories; even with the

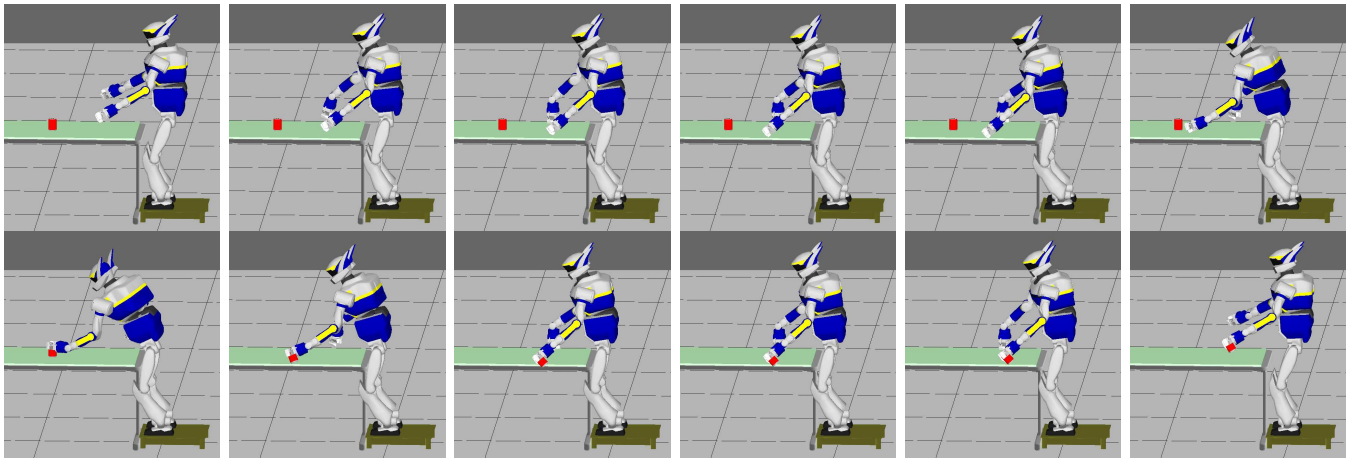

Fig. 1. Simulation results of the planner.

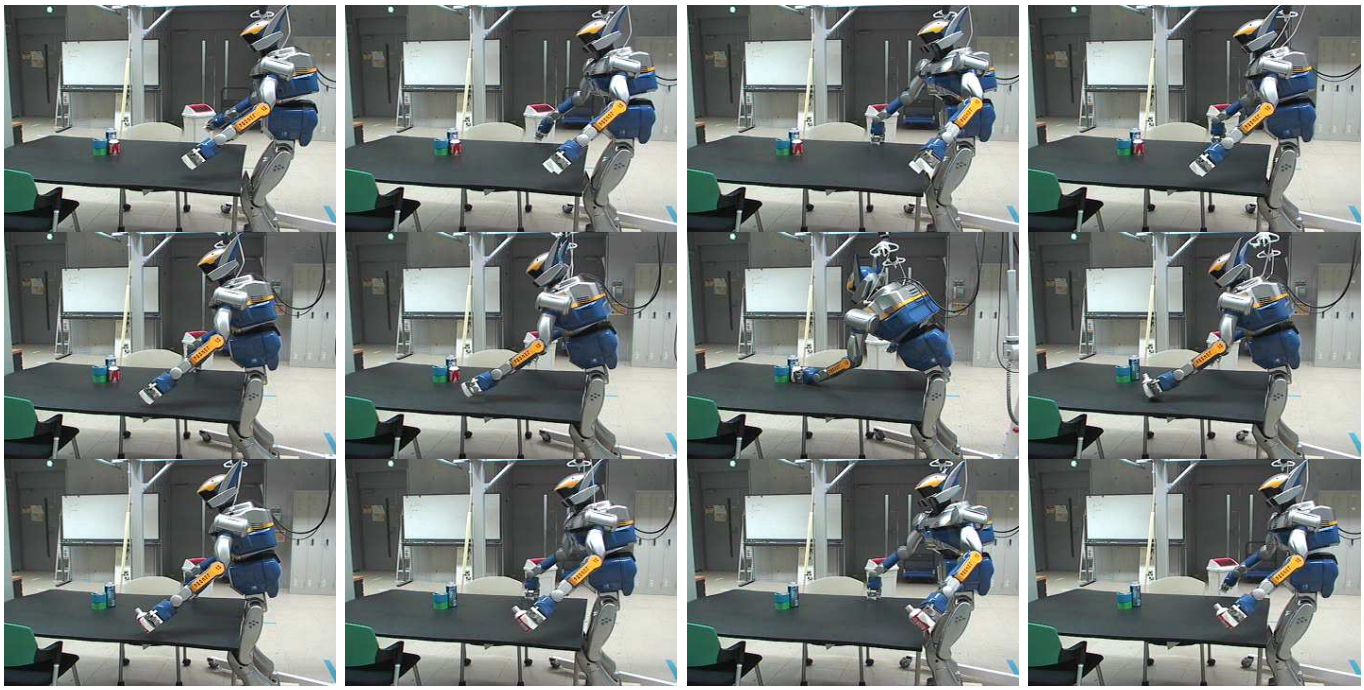

Fig. 2. Experimental results with HRP-2.

unavoidable light discrepancies, the HRP-2 realized perfectly the grasping of the can in several trials.

C. Discussion

It is obvious, that the illustrations are those obtained from a selected driven configuration and we intentionally limited the planner to not run forward all the possible configurations. It is easy to understand that, even for this simple example, the combinatorial possibilities would explode if all possible contacts and breaks are allowed to occur. Indeed the robot would also be allowed to put its feet and climb the table to reach the goal. This is a further implementation we are considering with various other examples. But, all the modules and functions are undergoing updates with more refined implementations taking into consideration additional features (such as the context, learning process, etc.) to filter and drive the solutions to plausible and limited case evolution.

We aim at realizing non-gaited motion through contact switching. We also allow contacts to occur on any part of the robot. Even though the experiments went quit well, it will nevertheless not be possible to have similar performance in more complex scenarios. We emphasize that a precision in the knowledge of the environment is needed at this stage. In order to allow actual humanoids using of contact points as supports, it is necessary to fulfill several hardware and software requirements.

Hardware issues consist mainly in (i) acquiring knowledge on contacts formation or break by means of an artificial haptic

sensing functionality, and (ii) having compliance at the contact points.

In order to be able to recover from unavoidable discrepancies and uncertainties in the planned trajectories, it is important to detect contact formation and breaking when they occur. This detection is necessary for many reasons. First, it is important for the robot to confirm that a contact is really made. Secondly, detecting the contact allows recovering from discrepancies and adapting the trajectories accordingly. Thirdly, the contact configuration tracking is useful to reduce internal efforts that may be engendered when it is not possible to detect contacts. Up-to-now, haptic sensing technology is not mature enough to allow this functionality on humanoids. We investigated various technologies, some of which are very promising. But, no technology is actually ready for a quick porting in the humanoid context. Moreover, most approaches are inspired from artificial skin sensing whereas what is needed is a functional aspect of haptic sensing. Therefore one may use a given technology for bina! ry detection, another one for localization and contact force measurement...

Compliance can be achieved in two ways: joint compliance or cover compliance. Many researchers seem to favor joint compliance. This is understandable because it is possible to instrument joints to measure its compliance. It is also possible to create artificial compliance from force sensing or by reducing controller PD gains. Of course, this is made mainly to reduce impact forces when the contact is made. But, in our case study, joint compliance is not enough. Indeed, what is needed is that the robot takes supports on the environment with any of its part. Hence, the contact should be 'stable'. A joint compliance will not allow contact area to spread or adapt to the environment local form to strengthen the support. In the other hand, this implies that the robot cover must be deformable. In this case, the problem is even more complex since it will not always be possible to know the exact location of contacts and the exact kinematics and dynamics models. Yet, these are still op! en problems in humanoid research because it concern a more general problematic of human/humanoid interaction.

The hardest problems (among others) we are still actively dealing with in algorithmic and software implementation are: (i) how to optimize the choice of the contact spots on a given part of the body and the possible supports? (ii) how to limit the choice and prioritize some parts among others? (iii) how to make the choice between creating a contact or breaking an existing one? (iv) if the choice is to break a contact, then, which one?

V. Conclusion and further work

A contact-support and motion planner for poly-articulated robots is presented. The originality of this planner is in allowing contact support to occur on any parts of the body and the environment. Although preliminary, simulation and experimental results show that the proposed methodology is viable. Given a target objective in terms of tasks, the planner generates a sequence of contact-switches and states for which an intermediary motion is generated. The objects in the environment are considered as obstacles but also as potential supports. There is still much work to do in refining all the developed modules to drive the sequence generation and lower the combinatorial tree complexity. We are also working on formulation issues in a context of more complex scenario.

Acknowledgment

Authors are thankfull to: Dr Pierre-Brice Wieber from INRIA for his explanations on HuMAnS and FSQP; and to Dr Jean-Paul Laumond from LAAS for his advices on this work. This work has been conducted at the AIST/CNRS Joint Japanese-French Robotics Laboratory (JRL) at ISRI/AIST, Tsukuba, Japan. Authors are also grateful to Dr Kazuhito Yokoi, Co-Director of JRL Japan and to all other JRL Japan members for various discussions on this work. The first author is supported by grants from the ImmerSence EU CEC project, Contract No. 27141 (FET-Presence) under the 6th Research program, and last author is supported from JSPS grant.

References

[1] K. Harada, H. Hirukawa, F. Kanehiro, K. Fujiwara, K. Kaneko, S. Kajita, and M. Nakamura, "Dynamical balance of a humanoid robot grasping and environment," in *IEEE International Conference on Robotics and Automation*, 2004.

[2] J.-C. Latombe, *Robot motion planning*. Kluwer Academic Publishers, 1991.

[3] H. Choset, K. M. Lynch, S. Hutchinson, G. Kantor, W. Burgard, L. E. Kavraki, and S. Thrun, *Principles of robot motion: theory, algorithms, and implementation*. MIT Press, 2005.

[4] K. Yamane, J. J. Kuffner, and J. K. Hodgins, "Synthesizing animations of human manipulation tasks," *ACM Transactions on Graphics, SIGGRAPH*, 2004.

[5] J. Xiao and X. Ji, "A divide-and-merge approach to automatic generation of contact states and planning of contact motion," in *IEEE International Conference on Robotics and Automation*, 2000.

[6] T. Siméon, J. Cortès, J.-P. Laumond, and A. Sahbani, "Manipulation planning with probabilistic roadmaps," *The International Journal of Robotics Research*, 2004.

[7] K. Hauser, T. Bretl, and J.-C. Latombe, "Non-gaited humanoid locomotion planning," in *IEEE/RSJ International Conference on Humanoid Robots*, 2005.

[8] K. Kaneko, F. Kanehiro, S. Kajita, H. Hirukawa, T. Kawasaki, M. Hirata, K. Akachi, and T. Isozumi, "Humanoid robot HRP-2," in *IEEE International Conference on Robotics and Automation*, 2004.

[9] J. Kuffner, K. Nishiwaki, S. Kagami, Y. Kuniyoshi, M. Inaba, and H. Inoue, "Self-collision detection and prevention for humanoid robots," in *IEEE International Conference on Robotics and Automation*, 2002.

[10] P.-B. Wieber, "On the stability of walking systems," in *The third IARP International Workshop on Humanoid and Human Friendly Robotics*, 2002.