Agimus: a new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers
Alexis Nicolin, Joseph Mirabel, Sébastien Boria, Olivier Stasse, Florent Lamiraux

To cite this version:
Alexis Nicolin, Joseph Mirabel, Sébastien Boria, Olivier Stasse, Florent Lamiraux. Agimus: a new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers. IEEE/SICE International Symposium on System Integration, Jan 2020, Honolulu, United States. hal-02466543

HAL Id: hal-02466543
https://hal.laas.fr/hal-02466543
Submitted on 4 Feb 2020

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.
Agimus: a new framework for mapping manipulation motion plans to sequences of hierarchical task-based controllers

A. Nicolin†, J. Mirabel†, S. Boria*, O. Stasse† and F. Lamiraux†

*AIRBUS, DVR, Toulouse, France
†LAAS-CNRS, Université de Toulouse, France

Abstract—In this paper we present the integration of a manipulation motion planner involving multiple contacts with an automated generator of controllers to execute the manipulation tasks. The novelty of the method is not only to produce a configuration space trajectory but also automatically formulate the controllers that perform the tasks while keeping balance on a humanoid robot. We demonstrate this approach fully integrated on a real Talos humanoid robot while using controllers formulated as a sequence of hierarchical Stack-of-Tasks.

I. INTRODUCTION

Manipulation planning is an important topic in humanoid robotics to achieve real world applications [1]. It is challenging since it involves constraints on the configurations of the robot and the objects: when an object is grasped, possibly by several grippers, its configuration depends on the configuration of the robot. When an object is not grasped, it should lie still in a stable equilibrium configuration. In addition, in humanoid robotics, constraints such as balance must also be taken into account. This problem is therefore hybrid, a discrete part on the constraints to activate, and a continuous part over the configuration space of the robot and the objects. Finding the constraints to activate is a difficult problem because several possibilities exist. For instance manipulation may involve walking steps with the robot [2]. Evaluating each possibility implies solving constraints from a candidate configuration where the constraints are not fullfilled, but where the distance to the target is shorter. Then, we project this configuration on the manifold of the constraints of the planning state the robot is currently in. This approach is for instance used in [3]. It demonstrates a HRP-3 humanoid robot picking up boxes from the ground and stacking them on a table. In order to speed up this step, convergence towards the submanifold may be done from a set of predefined stable poses instead of from a random configuration, such as in [4]. It is used to plan motions with articulated objects. It is also possible to add information by using a graph structure to mitigate with this hybrid approach. For instance in [5] a multi modal graph is used to switch between three states: reaching, walking and pushing. This approach is used to make Asimo manipulate a box from an initial configuration to a goal configuration using visual feedback.

In works such as [6, 7], the authors formulate a non linear optimisation problem over a short horizon using the whole dynamical model of the robot and the models of its contacts. This problem is then solved using Differential Dynamic Programming. This solution is currently very computationnally intensive and is therefore not suitable for a discovery phase or over a large time horizon. It might be however interesting during a refinement phase after having fulfilled the geometric constraints.

Finally a recent extension of MoveIt has been proposed to plan motions using tasks [8]. However this approach seems to focus more on planning of the symbolic parts rather than on the interaction between the various constraints. It shows however that this is a difficult problem as it is only a recent extension despite the impressive demonstrations achieved with MoveIt.

A. Contributions

In this paper, we propose a new framework to plan and execute manipulation tasks on a real humanoid robot with poor kinematic calibration. Figure 2 depicts the overall architecture. The pipeline is composed of three main steps. The first step (Section III) consists in estimating the configuration of the system (position of the table, position of the object). The second step (Section IV) consists in planning a manipulation path from an initial configuration to a goal configuration of the system. The third step (Section V) consists in mapping a reference manipulation path into a sequence of task-based hierarchical controllers. All these...
They are described in this paper. Supervisor handles synchronisation between the different nodes. Vision detects and publishes the pose of AprilTags.

Fig. 2. ROS Architecture: the 3 main nodes are estimation, planning and control. They are described in this paper.

A. Constraint graph

Fig. 3. Part of the constraint graph. Circular states are defined by sets of grasps. The hexagonal boxes correspond to waypoint states called pregrasps which goal is to ease the grasp of objects. Transitions connect reachable states between each other and contain contraints related to the connection paths. Figure 1 shows a configuration in state $g_2, h_4$ going to $g_1, h_3$ with a possible transition. The constraint graph contains 17 states. Figure 3 displays a part of the constraint graph corresponding to Talos robot manipulating a box.

B) State: a state is defined by specifying which handle is grasped by each gripper. It is therefore represented by a pair of grasps $((g_1, H_i), (g_2, H_j))$ where $g_1$ and $g_2$ are the robot grippers, and $H_i, H_j$ take values in $\{h_1, h_2, h_3, h_4, \text{None}\}$. $H_i = \text{None}, i = 1, 2$ means that gripper $i$ does not grasp anything. Thus the maximum number of states is 25. However we only consider a smaller number of states, since some combinations of grasps are always in collision. At the current time, those combinations are detected by a human operator, but this detection might be done automatically quite easily. After this pruning operation, the graph contains 17 states. Figure 3 displays a part of the constraint graph corresponding to Talos robot manipulating a box.

C) Transition: states are connected by edges called transitions. Transitions may contain additional constraints. For instance the loop transition of state $((g_1, \text{None}), (g_2, \text{None}))$ onto itself contains a constraint that locks the degrees of freedom of the box. Setting the box lies in a stable equilibrium is indeed not equivalent to the box not moving on the table.

D) Waypoint states: grasp configurations are by definition very close to collision. To improve efficiency of manipulation planning, we define waypoint states. Those states belong to transitions and force the path to go through intermediate states that help avoiding collisions. Figure 3 displays a configuration in a waypoint state that makes the left gripper less likely to be in collision while grasping.

III. STATE ESTIMATION

Before planning a manipulation motion, we need to know the initial configuration of the system. The robot and table are assumed to stand on a flat ground. The position of the object with respect to the robot camera is computed using AprilTags stuck on the object. Due to calibration and measure errors, the measured position of the object is never in contact with the surface of the table. It is always a few millimeters above or under the expected plane. The configuration of the system therefore does not belong to any state and the manipulation planning algorithm is unable to plan a path. To overcome this issue, we project the configuration as measured by sensors (AprilTag and joint encoders) onto state $((g_1, \text{None}), (g_2, \text{None}))$ using Newton-Raphson method as described in II.B.

Due to calibration errors, the projected configuration of the whole system is sometimes rather far away from the
Maintaining the robot CoM in the support polygon

Right Foot position
Left Foot position
Right gripper torque
Left Hand position
Right Hand position
The box must be in contact using the
Left gripper torque
Right gripper torque
The box must be in contact using the i-th handle
Robot configuration

measured configuration. As a result the robot fails to grasp
the object. To solve this problem, we decided to add
the position of the table in the configuration vector and to stick
some AprilTags on the table. The position of the table is
now also measured using computer vision and modified by
the projection on the initial state.

IV. MANIPULATION PLANNING

Once the initial configuration is estimated and projected
onto the initial state of the constraint graph, we define the
goal configuration by flipping the object upside down and
by keeping the same configuration for the robot. To solve
the manipulation planning problem, we run a variant of
Manipulation-RRT defined in [9] Section III. Manipulation-
RRT is a manipulation planning version of RRT that explores
the states of the constraint graph as follows:

1) Draw a random configuration $q_{rand}$.
2) Find the closest node $q_{near}$ in the current roadmap,
3) Find the state of this node in the constraint graph,
4) Sample a transition getting out of this state,
5) Extend $q_{near}$ along the transition up to $q_{new}$,
6) Try to connect $q_{new}$ to other connected components
of the roadmap.

The result is then optimised using the random shortcut
method. The output of manipulation planning is a sequence
of paths linking the initial and goal configurations. Each
segment of the sequence lies in a transition of the constraint
graph where it is subject to the constraints attached to this
transition. In the following section, we build a controller for
each segment in order to control the motion of the robot.

V. MAPPING A MANIPULATION PATH TO A SEQUENCE OF
CONTROLLERS

In this section, we explain how the manipulation path
planned in the previous section is mapped to a sequence of
task-based hierarchical controllers. The robot is controlled
by a software called Stack-of-Tasks [12] or SoT. The same
approach can be applied to other control architecture such as
OpenSoT [13].

A. Stack-of-Tasks

Like several other current control implementations, the
SoT is solving a hierarchy of Quadratic Programs to generate
an instantaneous whole body control. Different versions are
available, including an effort control version, but in the
current paper, a kinematic controller is used. Tasks are very
similar to the constraints described in Table I and therefore
share a large portion of code with the planning software.
Although planning provides trajectories avoiding joint limits
and self collision, it might happen that such limits are quite
close to the planned path. Then, if some perturbations
occur during control it becomes necessary to ensure that the
boundaries are respected. For this reason, the SoT enforces
priority levels. In a nutshell, the SoT solves iteratively this
set of equations:

\[
\begin{align*}
\dot{q}_{i+1} &= \dot{q}_i + (J_{i+1}P_i)\dot{T}_{i+1} + J_{i+1}\dot{q}_i + \frac{\delta T_{i+1}}{\delta t} \\
P_{i+1} &= P_i - (J_{i+1}P_i)(J_{i+1}P_i) \quad (2a) \\
T_i &= -\lambda_i T_i, \quad q_0 = 0, \quad P_0 = I.
\end{align*}
\]

\[a)\] Computation cost: the most time-consuming part in
the evaluation of eq. (2a) and eq. (2b) is the computation
of the pseudo-inverse. It is more efficient [14] to work in a
basis of the previous control task null space. Therefore at
iteration $i + 1$ the null space of the previous control task is
given by a basis $K_i$ such that $J_iK_i = 0$ and $K_i^TK_i = I$,
then using a SVD decomposition:

\[J_{i+1}K_i = [U_{i+1} V_{i+1}] [S_{i+1} 0 0] \begin{bmatrix} \frac{Y^{T}_{i+1}}{Z^{T}_{i+1}} \end{bmatrix} = U_{i+1}S_{i+1}Y^{T}_{i+1} \quad (3)\]

Then the null space of $J_{i+1}K_i$ is given by $Z^{T}_{i+1}$. Therefore
any new control $u_{i+1}$ is such that:

\[\dot{q}_{i+1} = \dot{q}_i + K_iu_{i+1} \quad (4)\]

not to perturbate $\dot{q}_i$. Therefore we slightly rewrite the control
eq. (2a) and the update eq. (2b) as:

\[
\begin{align*}
\dot{q}_{i+1} &= \dot{q}_i + (J_{i+1}K_i)\dot{T}_{i+1} + J_{i+1}\dot{q}_i + \frac{\delta T_{i+1}}{\delta t} \\
K_{i+1} &= K_iZ_{i+1}
\end{align*} \quad (5a) \quad (5b)
\]

Updating $K_{i+1}$ is simpler than computing $P_{i+1}$ and the
number of columns of $K_i$ decreases after each iteration. This
allows us to have this decomposition working at 1 kHz in
Talos robot.

B. Generation of the controllers

We again use the notion of constraint graph. We define a
Stack-of-Tasks for each transition in the constraint graph.
This set of tasks defines the desired control to apply for all
trajectories of this transition. For instance, when an object is
grasped, one task will control the gripper force around some
value so that the object does not slip.

Algorithm 1 in [9] automatically generates a graph of
constraints, for manipulation planning. For all the controllers,
we set as the highest priority task a task ensuring the robot
balance. The lowest priority task is always a tracking of
the configuration along the reference trajectory given by
the planner. To automatically generate the intermediate tasks, we

\begin{table}
\begin{tabular}{|c|c|}
\hline
$g_{CoM}(C_{rob})$ & Maintaining the robot CoM in the support polygon \\
$g_{RF}(C_{rob})$ & Right Foot position \\
$g_{LF}(C_{rob})$ & Left Foot position \\
$g_{RH}(C_{rob})$ & Right Hand position \\
$g_{LR}(C_{rob})$ & Left Hand position \\
$g_{grip}$ & Right gripper torque \\
$g_{grip}$ & Left gripper torque \\
$g_{box}$ & The box must be in contact using the i-th handle \\
$g_{posture}$ & Robot configuration \\
\hline
\end{tabular}
\caption{Set of constraints handled in the control process}
\end{table}
We briefly describe a system extension allowing to use the overall architecture on robot supporting the ros_control [15], architecture. From a system integration point of view, we developed a software extension turning the Stack-of-Tasks into a controller in the ros_control architecture. The ros_control library is providing a common API for controllers and robot hardware. The same controller can be used over several hardwares, and several controllers can be tested on the same hardware.

This has several advantages because it is possible to use either the balance controller provided by the robot manufacturer PAL-robotics or our own balance control software. The second advantage is provided by the Gazebo simulator support of ros_control. Thus the same control software can be used either in simulation or on the real robot. Using ros_control simplifies the software maintenance as basic blocks such as the dynamics (Pinocchio Dynamics in Fig.5), the solver, and the tasks can be reused in other robots as well.

The current limitation of ros_control is its lack of composability with respect to actuators which have a non classical set of sensors (i.e. encoders). For instance the Talos robot is equipped with temperature and torque sensor on each actuator. For this reason a modified version of the API must be used on the robot.

VI. EXPERIMENTS

A. Description of the robot

The experiments are done on a Talos robot [16]. This robot has a height of 1.75 m, two torso joints, and two 7 DOFs arms. Grippers and the kinematics were designed to maximise manipulability in front of the robot. The vision system is an RBG-D Orbbec Astra camera. The robot has two computers: two i7 2.7 GHz, one for control and one for vision and planning. The control computer is running a Linux kernel patched with the RT_PREEMPT extension. The Manipulation-RRT algorithm is run on the second PC of the robot as well as the vision system based on ViSP [17]. The second PC runs a normal linux kernel on a 16.04 Ubuntu.

B. Inputs of the problem

The problem is the one described in [16] a Talos robot has to turn a box lying on a table upside-down using its two grippers. The inputs given to the algorithms are the kinematic models of the robot (and its grippers), the box (and its handles’ positions) and the environment (i.e. the table),
and also a goal configuration, which is here to have the box upside-down in relation to its initial pose. Finally, the robot computes the initial state of the world (its configuration and the position of both the table and the box) and the sequence of movements needed to perform the task.

C. Flow of computation

a) Initialisation: In planning, control and estimation, a Python script generates the constraint graph for the grippers and handles. To each transition in planning corresponds one controller in control. Each state in estimation has its equivalent in planning. The intermediate waypoint gives an overall structure to the motion and makes it easier to tackle the motion planning problem and the control problem. The graph includes CoM constraint to maintain balance and a visibility constraint on the box.

b) Runtime: The estimation node first estimates the robot configuration, the poses of the table and the box and the current state in the constraint graph, from various inputs (AprilTags, encoders).

This estimation is sent to the planning node, in order to find a trajectory from the current world state to the goal configuration, using the constraint graph. The output trajectory is a sequence of elementary paths. An elementary path is a path that belongs to a single transition of the constraint graph. During the estimation and planning phase, the robot is standing still.

This sequence of elementary paths is then used as reference by the control node. This node switches between controllers according to the transition each elementary path belongs to.

Both the planning and estimation nodes are implemented using the Humanoid Path Planner library[18], or HPP. The control node is implemented using Stack-of-Tasks.

D. Typical solution

As the motion planner is based on RRT there is no guarantee that the robot will find the same solution from one instance of problem to another. The constraints and the intermediate waypoints however provide a way to structure the solution found by the robot. A typical solution is provided in Figure 6. In this solution the robot finds the sequence of contacts to perform in order to swap the box. This sequence of contacts is implemented through a sequence of constraints. Each set of constraints corresponds to a specific set of tasks. The reference to be given to the robot is computed from the trajectory found by the motion planner. The corresponding Stack-of-Tasks are created through automatically generated python code interpreted by the second onboard computer of the Talos robot. The reference posture trajectories, and the reference end effectors poses are also sent to the SoT by the planner.

E. Discussion

In its current form the system does not track failures. For instance it happens that in some configurations the box slides from the robot gripper. This has been partly fixed by adding rubber to the grippers to increase the friction coefficient while holding the object. Detecting failure with vision would allow the robot to detect that the plan could not be fulfilled. However, recovery from failure may lead to complex solutions which are not feasible. For example if the box fall on the floor, the situation involves a more complex motion such as kneeling down. Finding this would involve too computationally intensive searches for the current system. In this case, it might be easier to avoid such failures or ask for the help of an operator. To help with this, we can add waypoints in the plan where the robot has to check if
everything is where the plan tells it should be (i.e. box in hand or box on the table).

We first tried to run the path planned in Section [IV] in open loop. This attempt failed due to an error of the relative position between the gripper and the object of up to 2 centimeters. This error is probably due to

1) a poor calibration of the kinematic chain,
2) the high number of joints in the actuation chain: 9 between the camera and the end effector, 10 between the ground and the end effector.

Although we could spend time better calibrating the robot, we aim at achieving tasks with an accuracy that cannot be obtained only by calibration.

To reach this goal, we are currently attaching and calibrating the poses of AprilTags stuck to the robot end-effectors. In planning, we already constrain the object to be at the center of the image. This is conservative, but it increases the likelihood that

- the object is visible even in case of localisation errors,
- the gripper is visible when in pregrasp state.

Thus, in pregrasp state, we will be able to compute the relative pose of the gripper with respect to the object, using the camera. Then, by adding a visual servoing task during the pregrasp phases of the trajectory, we should be able to achieve our desired accuracy to grab the box.

VII. CONCLUSION

In this paper we have demonstrated the deployment of a solution integrating vision, planning and control on a full size humanoid robot. Based on high level information such as handles and constraints of the robot, the system is able to generate a sequence of controllers as well as their reference trajectories. The plan is then executed on the robot using the controllers.

ACKNOWLEDGMENTS

This work is supported by the cooperation agreement ROB4FAM.

REFERENCES