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://laas.hal.science/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.
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 fulfilled, 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 1 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 second step (Section V) consists in mapping a reference manipulation path into a sequence of task-based hierarchical controllers. All these
AprilTags. between the different nodes. Vision detects and publishes the pose of
control. They are described in this paper. Supervisor handles synchronisation
Fig. 2. ROS Architecture: the 3 main nodes are estimation, planning and
Control
Robot
Planning
Supervisor

configuration space of the table and of the box
1
of the robot,
C
180
on the top, rotated 180° from each other, to allow grasping from the front and from behind ; two handles on the bottom
of the object, to allow grasping even if the box is placed

steps use a common representation of the constraints the
system is subject to, described in Section II.

An example of manipulation problem that our method is
able to tackle is depicted in Fig. 4.

II. PROBLEM STATEMENT

Although our work is general and applies on instances of manipulation problems with several robots and several
objects, we illustrate our framework on a simpler example where a Talos robot is requested to grab a wooden box lying
on a table, turn it upside-down, then put it back on the table. The configuration space of the whole system is then:

\[ C = C_{rob} \times C_{table} \times C_{box} \]  

With \( C_{rob} = SE(3) \times \mathbb{R}^{32} \) is the configuration space of the robot, \( C_{table} = C_{box} = SE(3) \) are respectively the configuration space of the table and of the box.

Note that although the table is static, its position needs to be estimated using computer vision before planning and executing any motion.

A. Constraint graph

In this section, we use the same concepts and notations as in [2].

\( SE(3) \) is the group of rigid-body transformations.

a) Gripper, handle, and grasp: the robot is equipped with 2 grippers represented by frames attached to each end
effector. To each object, we associate one or several frames called handles that specify how the object can be grasped: a
grip corresponds to a gripper frame being superposed with a handle frame. In example of Figure 1, 4 handles denoted
by \((h_1,h_2,h_3,h_4)\) are attached to the object ; two handles on the top, rotated 180° from each other, to allow grasping
from the front and from behind ; two handles on the bottom
of the object, to allow grasping even if the box is placed

b) State: a state is defined by specifying which handle is grasped by each gripper. It is therefore represented by a pair of pairs \((g_1,H_1),(g_2,H_2)\) where \(g_1\) and \(g_2\) are the robot grippers, and \(H_1, H_2\) take values in \((h_1,h_2,h_3,h_4,None)\). \(H_i = None, i = 1,2\) 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 \((g_1,None),(g_2,None)\) onto itself contains a constraint
that locks the degrees of freedom of the box. Setting that
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 def-
inition 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 4 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 sytem. 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 config-

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
\((g_1,None),(g_2,None)\) using Newton-Raphson method as described in [11] Section II.B.

Due to calibration errors, the projected configuration of
the whole system is sometimes rather far away from the
TABLE I
SET OF CONSTRAINTS HANDLED IN THE CONTROL PROCESS

<table>
<thead>
<tr>
<th>Constraint</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$gC_{rob}$</td>
<td>Maintaining the robot CoM in the support polygon</td>
</tr>
<tr>
<td>$gRF(C_{rob})$</td>
<td>Right Foot position</td>
</tr>
<tr>
<td>$gLF(C_{rob})$</td>
<td>Left Foot position</td>
</tr>
<tr>
<td>$gRH/h(C_{rob})$</td>
<td>Right Hand position</td>
</tr>
<tr>
<td>$gLH/h(C_{rob})$</td>
<td>Left Hand position</td>
</tr>
<tr>
<td>$g_{\tau}(C_{rob})$</td>
<td>Right gripper torque</td>
</tr>
<tr>
<td>$g_{\tau}<em>L(C</em>{rob})$</td>
<td>Left gripper torque</td>
</tr>
<tr>
<td>$g_{\tau}Box(C_{Box})$</td>
<td>The box must be in contact using the i-th handle with i $\in {1, \cdots, 4}$</td>
</tr>
</tbody>
</table>

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 $q_{\text{rand}}$.
2) Find the closest node $q_{\text{near}}$ 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 $q_{\text{near}}$ along the transition up to $q_{\text{new}}$.
6) Try to connect $q_{\text{new}}$ 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 operates iteratively this set of equations:

$$
\dot{q}_{i+1} = \dot{q}_i + (J_{i+1}P_i)^T(T_{i+1} - J_{i+1}\dot{q}_i + \frac{\delta T_{i+1}}{\delta t})
$$

$$
P_{i+1} = P_i - (J_{i+1}P_i)^T(J_{i+1}P_i)
$$

with $i \in \{1, \cdots, n\}$, $q \in C_{rob}$ the configuration vector of the robot, $\dot{q}$ the related velocity, $T_i$ the i-th task, $J_i = \frac{\delta T_i}{\delta q}$ its Jacobian, $\lambda_i$ its priority, $\dot{q}_0 = 0$, $P_0 = I$.

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 $K_i^T K_i = I$, then using a SVD decomposition:

$$
J_{i+1}K_i = [U_{i+1} V_{i+1}] \begin{bmatrix}
S_{i+1} & 0 \\
0 & 0
\end{bmatrix} \begin{bmatrix} Z_{i+1}^T \\
Y_{i+1}^T
\end{bmatrix} = U_{i+1} S_{i+1} Y_{i+1}^T
$$

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

$$
\dot{q}_{i+1} = \dot{q}_i + K_i u_{i+1}
$$

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

$$
\dot{q}_{i+1} = \dot{q}_i + (J_{i+1}K_i)^T(T_{i+1} - J_{i+1}\dot{q}_i + \frac{\delta T_{i+1}}{\delta t})
$$

$$
K_{i+1} = K_i Z_{i+1}
$$

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
change the function MAKETRANSITIONS. MAKETRANSITIONS builds controllers as follows. It takes as input two states $S_0$ and $S_1$. $S_1$ corresponds to the state $S_0$ with exactly one additional grasp (i.e. association between a gripper and a handle). As in planning, grasping is divided into two steps:

1) From $S_0$, i.e. only existing grasps, to a pregrasp waypoint: no special control should be done for this grasp. One should only maintain the grasps which are already in $S_0$.

2) From the pregrasp waypoint to $S_1$: at this moment, only a contact detection task, using torque sensors of the robot, is added. In the future, we also want to add a visual servoing task to improve our system success rate.

Eventually, when there is a contact break, two additional steps are added. One for the contact break, until some preplacement pose and another for the motions far from placement. A specific task for contact break is added in the former step, to register the object position relative to the hand placement. A specific task for contact break is added in the pregrasp state.

The Stack-of-Tasks provides a framework that computes a task for contact break, until some preplacement pose and another for the motions far from placement. A specific task for contact break is added in the pregrasp state.

Eventually, when there is a contact break, two additional steps are added. One for the contact break, until some preplacement pose and another for the motions far from placement. A specific task for contact break is added in the former step, to register the object position relative to the hand placement. A specific task for contact break is added in the pregrasp state.

The Stack-of-Tasks provides a framework that computes a task for contact break, until some preplacement pose and another for the motions far from placement. A specific task for contact break is added in the former step, to register the object position relative to the hand placement. A specific task for contact break is added in the pregrasp state.

Fig. 4. Sequence of controllers. Portion of path where the robot holds the object with the left hand and grasps it with both hands, passing by a pregrasp state.

Fig. 5. Interaction between $SoT$ and $HPP$. ROS is used for its easily accessible data exchange system and future addition of video to perform visual servoing.
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 reference by the control node. This node switches between controllers according to the transition each elementary path involves. 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 SoT the Talos robot. The reference posture trajectories, and the reference end effectors poses are also sent to the SoT by the planner.

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 SoT 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


