A Hybrid Control Architecture for Mobile Manipulation*

L. Petersson⋆, M. Egerstedt†, and H.I. Christensen⋆

⋆{larsp,hic}@nada.kth.se
NADA/CVAP
Royal Institute of Technology
SE - 100 44 Stockholm, Sweden

†magnuse@math.kth.se
Optimization and Systems Theory
Royal Institute of Technology
SE - 100 44 Stockholm, Sweden

Abstract

In this article we present a scheme for mobile manipulation by introducing a Mobile Manipulation Control Architecture (MMCA). This architecture is motivated by a need for a systematic control structure for robotic manipulation within a behavior based framework. The control structure enables integration of the manipulator into a behavior based control structure for the platform. Furthermore, our suggested MMCA is designed in such a way that it supports design and performance analysis from both a manipulator dynamics and a hybrid automata perspective.

1 Introduction

In many autonomous robot applications the ability to perform *mobile manipulation* is of key importance. These applications range from robots in space or under water to construction and service robotics. This last category is the motivating application for this work and we believe that there is much to gain in terms of performance if an intelligent service agent is not restricted to manipulating its environment statically, with a fixed base.

The reason for this is basically three fold. First of all, the speed of the manipulation task can be increased if the manipulation is done with a moving base. If, for example, the task is to pick up an object such as a cup on a table, it seems desirable to have the robot preparing for the grasping motion by placing the arm in a suitable position while approaching the object. Secondly, when opening a door (inwards, towards the robot) the base has to move to avoid getting hit by the door. In this case, the base has to move simultaneously with the arm in order for it to execute the desired task successfully. The third argument for mobile manipulation is a structural one. If the robotic system is structured in a *behavior based* way, which is the case in this work, the idea is that many different behaviors, responses to sensory inputs, might be active simultaneously. If this is to be the case while performing manipulation, other behaviors than the specific manipulation behavior could be active which would mean that the robot must be able to move its base while performing its manipulation tasks, which calls for a coordinated arm and base motion. This argument implies not only that we need mobile manipulation but also that the manipulation has to be done in a systematic way that allows for us to use it within a behavior based framework. In this article we propose a solution to this architectural problem that we will refer to as a Mobile Manipulation Control Architecture (*MMCA*).

The subject of mobile manipulation in a dynamic environment is still in its early stages unlike the static counterpart which has been thoroughly investigated for many years [1, 6, 4]. The reason why so little has been done in this field is probably that the hardware, i.e. a moving platform with a manipulator, has not been available. Nowadays, however, such systems are readily available at a reasonable cost. To investigate some of the different issues arising in mobile manipulation, such as arm/base coordination, internal motion of the platform during manipulation, unintentional motions of the platform due to arm induced oscillations and path planning questions, the *Centre for Autonomous Systems* (CAS) in Stockholm has chosen to work with a Nomadic Technologies XR4000 base platform together with a Puma560 manipulator arm, as seen in Figure 1.

What will be investigated in this article is thus how mobile manipulation should be structured so that the

*This work was sponsored in part by the Swedish Foundation for Strategic Research through its Centre for Autonomous Systems at KTH.

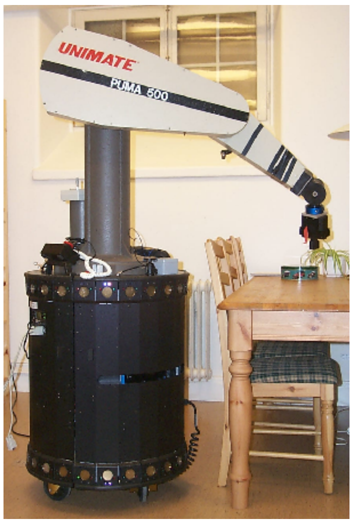


Figure 1: The Nomad XR4000 platform with a Puma560 arm on top on which the MMCA is implemented.

robotic system is coherent with a behavior based architecture while at the same time allowing for a theoretical treatment of the properties of the system. This will be done by modeling the different *manipulation primitives* as nodes in a *hybrid automaton* running in parallel with a behavior based system, acting on the base platform.

The outline of this article is as follows: We first discuss, briefly, the type of structural considerations that a MMCA has to respect if the mobile manipulation is to be conducted within a behavior based framework. We then present our MMCA followed by a discussion about system specification and implementation issues, in Section 3 and 4 respectively. We conclude, in Section 5, with some preliminary experimental results.

2 Behavior Based Mobile Manipulation

2.1 Behavior Based Robotics

For autonomous robots operating in an unknown, dynamic environment, a successful way of structuring the controllers is within a behavior based framework [3]. This modular approach has the nice structural property that it allows for decentralized controllers to be used, dedicated to doing different tasks such as door traversal or obstacle avoidance. Therefore a control architecture for mobile manipulation would benefit from being consistent with a behavior based control structure.

However, questions concerning safety, stability or optimality are hard to address when behaviors are fused or switched between and this is a complication that becomes serious when doing mobile manipulation. For slow robots with stable dynamics, such as the Nomadic platform without an arm mounted on top, this complication is not of major importance. When controlling the manipulator arm, however, the ability to theoretically analyze and verify the behavior is crucial due to its very complex dynamics.

In this article, we identify different manipulation primitives such as approach-object or grasp-object and they are viewed as discrete nodes in a hybrid automaton. The idea is that a manipulator behavior, such as pick-up-object, is built up from a number of these primitives, with only one primitive active at a time. The reason for this restriction is that it allows us to view the manipulator as a hybrid automaton on which we can use the existing theories for safety, performance or stability verification [10, 12].

A hybrid automaton is formally considered to be a collection (Q, X, I, f, E) where Q and X are sets of discrete and continuous variables respectively. I is a set of initial conditions, f describe the continuous and E the discrete evolution of the states. A discrete state combined with the continuous dynamics connected to that state will be referred to as a *node* in the automaton. The general idea of this construction can be seen in Figure 2.

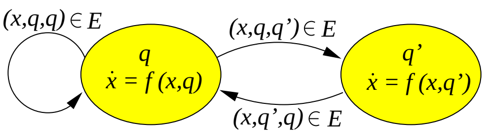


Figure 2: The basic structure of a hybrid automaton.

Although this article will not deal with hybrid automata theory *per se*, the reason for introducing it is that we want our MMCA to be designed in such a way that it allows for an analysis of the manipulator behaviors to be done within the hybrid automata framework. Therefore it needs to support a formulation of the structure in the above mentioned way.

2.2 Manipulator Dynamics

For a redundant, mobile manipulator like our platform, with 9 degrees of freedom for controlling the 6 dimensional end-effector position and orientation, it is possible to decompose the generalized joint forces, Γ, acting on the platform, into two decoupled vectors as

$$\Gamma = J^T(q)F + \big(I - J^T(q)\bar{J}^T(q)\big)\Gamma_0, \tag{1}$$

where q is the joint coordinates, and the Jacobian matrix, $J(q)$, relates the joint forces to the operational forces of the manipulator [13, 9]. In (1), $\bar{J}(q)$ is the so called dynamically consistent generalized inverse [8], and $(I - J^T(q)\bar{J}^T(q))$ is the projection operator onto $\mathcal{N}(J)$ (the null-space of J). This decoupling gives us means for controlling the end-effector dynamics by F at the same time as the internal motions can be controlled by Γ_0.

There are already a number of redundancy resolution schemes for manipulation [5], but this decoupling furthermore suggests means for "hiding" the observed, unintentional motions of the base given by the actions from other active base behaviors. This could be done in the internal motions of the mobile manipulator by a correction of both the base and the manipulator configuration made by a suitable choice of Γ_0. This is a desirable feature since it allows us to not only deal with singularity avoidance and energy minimization in the internal motions, but also provides us with a systematic tool for allowing for movements in the base that are not given by the current active manipulator behavior. Therefore the observed actions from the base behaviors can be compensated for by choosing actions that lay in the null-space of the Jacobian of the combined platform-arm system.

Using a hybrid automaton makes it possible to have different kinds of control algorithms specified individually within the nodes in the automaton for different situations that can arise during a manipulation task. Within each of these nodes it is thus possible to specify different redundancy resolution schemes. The nodes could also contain schemes for dealing with the disturbances caused by the platform behaviors by compensating for them by choosing appropriate counteractions. If, for instance, a compliant motion, like opening a door, is conducted and an obstacle avoidance behavior becomes active, this new action can be dealt with without affecting the position and orientation of the end-effector.

2.3 MMCA

Within the MMCA two distinct parts can be identified, corresponding to base and manipulator behaviors respectively. The part of the system that only governs the base platform directly is, as already mentioned, going to be a behavior based system. Different robot behaviors are identified, e.g. avoid-obstacle or move-to-goal, and their functionality is defined by a mapping from sensory data to a desired base action. These desired base actions are fused together by an arbitration mechanism, as seen in Figure 3. For instance, while approaching a target, an obstacle avoidance behavior has to be active for safety reasons while the performance is improved if the robot tries to approach the goal at the same time as it is avoiding obstacles [3, 2].

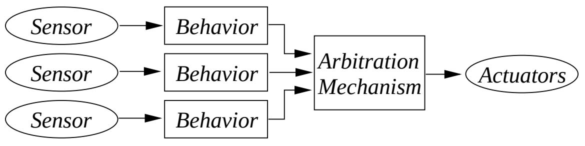


Figure 3: Block diagram of the behavior based control architecture.

The effect that the manipulator has on the base is just regarded as another behavior, entering the arbitration mechanism with another suggested base motion, as seen in Figure 4.

For the manipulation part of the MMCA, the questions are slightly different since the states of the mobile manipulator are going to be affected by not only the manipulation behavior, but also by other active base behaviors that are viewed as discrete disturbances due to the slow base dynamics.

Based on these structural considerations, given by the behavior based platform system, combined with a desire to come up with a system suitable for performance analysis, we choose, as already mentioned, to model the manipulation behaviors as hybrid automata. In these automata, the nodes correspond to the different manipulation primitives, such as grasp or approach-object, needed to accomplish a given task, and thus the core of the manipulation part of the MMCA is an engine that executes hybrid automata. Each automaton thus holds information about the manipulation primitives and how they are connected to each other.

The architecture is designed to be open and allows the user to experiment with the contents of the primitives freely, e.g. internal representations and algorithms, as long as the primitive contains:

(i) A function returning desired state (in our case joint angles and platform pose)

(ii) Conditions for when to make the discrete transitions

Finally, when applying this MMCA approach to a complex manipulation task, some basic design steps can be identified. First of all, break down the task

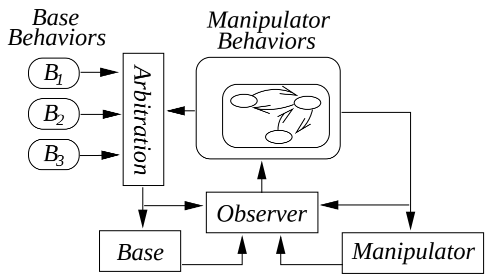


Figure 4: The hybrid automaton viewed as a behavior with a generalized sensory input that may also contain information about changes in the configuration of the automaton.

into sub-tasks where a single, dedicated, continuous controller can be used. Secondly, for every sub-task, write a primitive implementing the continuous controller and define conditions for when to switch between the sub-tasks. And finally, set up the definition of the hybrid automaton, i.e. define the discrete transitions etc.

3 System Specification

There is a need to define the automaton in a structured and systematic way, making it easy to reuse and reorder nodes in different configurations. Therefore, a special language (see below) has been developed for specification of systems.

A "program" written in this language begins to define which interface to use, i.e. which hardware server (section 4), and also the initial node. All the nodes in the automaton are then listed.

Program structure	Node structure
interface	*name*
initial node	*type*
nodes	*parameters*
	transitions

Each node is specified by name, type, parameters and transitions.

The transitions refer to the other nodes, or itself, by using the specified name. The type of a node determines the functionality, such as the controller, and also defines which parameters or initial values that can be passed to the node.

A sample file that defines the task of opening a door might look like:

```
INTERFACE = Puma560_XR4000;
INITNODE = Approach;
BEGIN
   NAME = Approach;
   TYPE = Visual_Servo;

   Object = Door_Handle;    % Servo on a door handle
   TRANSITION[End_Position] = Grasp;
END
BEGIN
   NAME = Grasp;
   TYPE = Grasp_Object;

   Object = Door_Handle;    % Grasp a door handle
   TRANSITION[Got_Grip]  = Pull;
   TRANSITION[Lost_Grip] = Approach;
END
BEGIN
   NAME = Pull;
   TYPE = Follow_Arc;

   Radius = 0.8;            % Estimate of the arc radius
   Angle  = 90;             % Open door 90 deg.
   TRANSITION[Ready] = End; % Terminates the control cycle
END
```

$$\begin{tikzcd}
\text{Approach} \arrow[r, bend left] & \text{Grasp} \arrow[l, bend left] \arrow[r, bend left] & \text{Pull}
\end{tikzcd}$$

Figure 5: The automaton defined in the example.

During initialization, a lexical, syntactic and semantic check is performed, ensuring that only a well defined automaton can be executed.

4 Implementation

Our MMCA was built on top of low level PD-controllers for the manipulator arm and the platform. This type of model independent control strategy has the advantage that it decouples the controllers from the specific hardware making it easier to upgrade or change equipment.

The implementation was made using C++ under the operating system QNX [14]. QNX was chosen because of the need for a real-time OS since we want to be able to control a system with a fixed and high (> 100Hz) sampling frequency such as the Puma arm. However, the behavior based system that controls the platform resides on a Linux system which is a non real-time OS.

The language described in the previous section was implemented using the well known lexical and parsing utilities `lex` and `yacc` [7, 11], which makes it easy to extend the grammar if needed.

A design goal has been to separate all hardware specific components from the rest of the system, increasing portability. All communication with low-level routines is made using a client-server approach. Specifically, the interface mentioned in section 3 is implemented by the user and registered by the system under a name to be used in the definition file. An advantage here is that it is possible to go from a simulated environment to the real hardware just by specifying another interface.

The MMCA-engine itself has no notion of which states the system has, neither does it know the specific functionality of the primitives, this is left to the user to choose. To achieve this, whenever the user wants to add a manipulation primitive to the system, it must inherit from a base class `Primitive` defining a common interface that has a few functions the user is required to implement. These are `jumpTo()` and `getState()`.

```
class Primitive
{
  public:
     Primitive();
     virtual ~Primitive();

     virtual int jumpTo(State *state, double time);
     virtual getState(State *state, double time);

                 ⋮

};
```

In every control cycle, `getState()` will return the desired state, in our case Puma joint angles and platform pose.

In order to define a proper representation of the system, the user has to inherit from a base class `State`. Our system setup can be described by

```
class JointState : public State
{
  public:
     JointState();
     ~JointState();

     double pumajoints[6];     % The puma has six d.o.f.
     int update_puma;

     double platformcoord[3]; % The platform has three d.o.f.
     int update_platform;
};
```

It is worth noting, that the above representation also keeps track of whether the data is updated or not since the last cycle. This makes it possible to calculate the desired platform pose at a lower frequency than the arm joints due to slower platform dynamics, thereby reducing the computational complexity.

The `jumpTo()` function returns an enumerated value (integer) depending on which jump-condition is triggered. A mapping of this value to a name is then used in the `TRANSITION[...]` section to specify where to jump (see section 3).

A complete sample program could be

```
void main()
{
   // Instantiate an automaton running at 100Hz
   MMCA automaton(100);

   // Read the definition file
   if(automaton.loadSet("hybrid.cfg")){
      cerr << "Error while reading config file!" << endl;
      exit(-1);
   }

   // Start the control loop. (Non-blocking)
   automaton.start();

   // If needed, add functionality to monitor and
   // reconfigure here.
   // There exists access functions to the running automaton.
}
```

Note that this implementation supports that more than one automaton is active at the same time, controlling other aspects of the system, or possibly other manipulator arms.

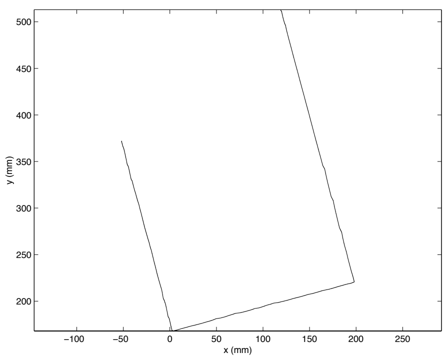
5 Experimental Results

The MMCA was used to implement a simple example showing how the end-effector can be kept stationary, while disturbing the platform motion manually, with a joy-stick. An experiment like this is relevant in the context of a behavior based system where unpredictable motion of the platform must be transparent to the manipulation task. Even though it must be considered a small and preliminary experiment, it is still relevant both for showing that our MMCA actually works, and that it solves the unintentional motion rejection problem.

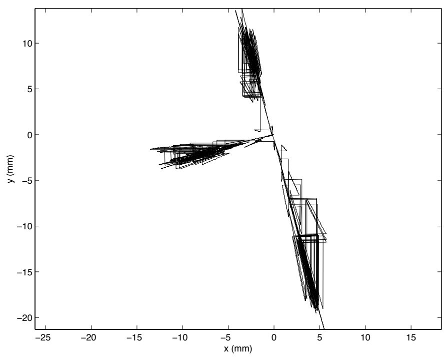
The only primitive, `Stay`, needed for accomplishing this manipulation task consists of an initialization part and a cycle part. In the initialization part, the transformation, T_0, from a fixed Cartesian reference coordinate system to the end-effector is calculated as $T_0 = T_1T_2T_3$. Here, T_1 is the transformation, corresponding to the platform pose, from the reference coordinate system to a base fixed system. T_2 is a constant, purely translative transformation corresponding to the height of the XR4000. Finally, the joint angles give the arm transformation, T_3, based on a forward kinematic arm model.

After this, `Stay` cycles $T_3 = (T_1T_2)^{-1}T_0$, where T_1 and T_2 are updated at each cycle. We then from T_3, based on the inverse arm kinematics, get the desired joint angles needed to reject the effect of the base motions on the end-effector.

An experimental result can be seen in Figures 6-7. It can be noted that the base motions produce deviations in the end-effector position of order of centimeters. The reason for this deviation is due to the time



(a) Base translation (mm)



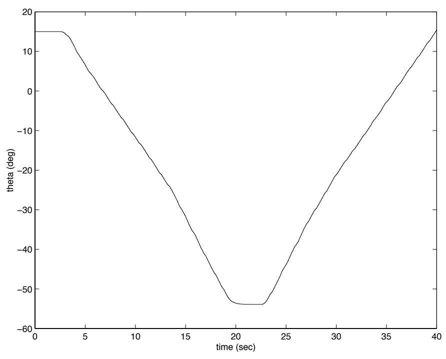
(b) End-effector position (mm)

Figure 6: Rejection of platform movements. In this example, the platform was purely translated. In the upper figure the platform motion is shown and in the lower the end-effector position is plotted, centered around its initial position. From b it is obvious that the three dominating modes correspond to phase lag for each motion segment.

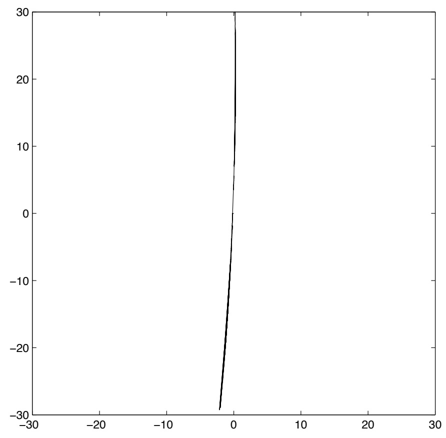
delay induced by the slow odometry update in the Nomad XR4000. This fact suggests that further research needs to be done in the area of state estimation.

6 Conclusions

In this article we presented a control architecture for mobile manipulation within a behavior based framework. Different manipulation primitives were identified and modeled as nodes in a hybrid automaton, resulting in a hybrid control system, suitable for performance verification and control design. This automaton was viewed as a manipulator behavior, acting on the platform together with other base behaviors.



(a) Base rotation (deg)



(b) End-effector position (mm)

Figure 7: Rejection of a rotational base motion (upper figure). In the lower figure, the end-effector position is displayed.

We implemented our MMCA on a Puma560 arm, mounted on a Nomad XR4000, and some preliminary experiments were conducted. We were able to use our MMCA to reject manually controlled base motions that could be thought of as actions from base behaviors. This indicates that our wish to treat the mobile manipulation problem from a system point of view works in practice as well as in theory.

Our solution did not impose any specific constraints on how to solve the detailed control problems and was therefore suitable as a research platform. It is our belief that this platform will be valuable when research in other areas is to be conducted. These areas could range from redundancy resolution schemes to visual servoing.

In the near future, the problem concerning state estimation needs to be addressed. This involves the construction of an observer that deals with both the stochastic time delays between the two parts of the MMCA, as well as sensor fusion issues. It is also our intention to show that we, by exploiting the MMCA, can produce a robust hybrid automaton for real, complex manipulation tasks.

Acknowledgment

The authors would like to thank Anders Orebäck for helpful discussions and system support.

References

[1] P. Allen, A. Timcenko, B. Yoshimi, and P. Michelman. Automated Tracking and Grasping of a Moving Object with a Robotic Hand-Eye System. In *Visual Servoing Workshop, IEEE Conference on Robotics and Automation*, 1994.

[2] M. Andersson, A. Orebäck and M. Lindström and H.I. Christensen. : An Intelligent Service Robot. In Intelligent Sensor Based Robot Systems *Lecture Notes in Artificial Intelligence*. Springer Verlag, Heidelberg, Germany, September 1999.

[3] R.C. Arkin: *Behavior-Based Robotics*. The MIT Press, Cambridge, Massachusetts, 1998.

[4] H. Bruyninckx and J. De Schutter: Invariant Hybrid Position/Force Control of a Velocity Controlled Robot with Compliant End-Effector Using Modal Decoupling. *International Journal of Robotics Research*, 16(3):340–356, 1997.

[5] B.E. Bishop and M.W. Spong: Control of Redundant Manipulators Using Logic-Based Switching. Proceedings of the *37th IEEE Conference on Decision and Control*, Tampa, Florida, USA, December 1998.

[6] P.I. Corke: *Visual Control of Robots: High-Performance Visual Servoing*. John Wiley and Sons Inc., CSIRO Division of Manufacturing Technology, Australia, 1996.

[7] S. C. Johnson: *Yacc: Yet Another Compiler Compiler*. Computing Science Technical Report No. 32, Bell Laboratories, Murray Hill, New Jersey, 1975.

[8] O. Khatib: Inertial Properties in Robotics Manipulation: An Object-Level Framework. *International Journal of Robotics Research*, vol. 14, no. 1, February 1995, pp. 19–36.

[9] O. Khatib, K.Yokoi, K.Chang, D.Ruspini, R.Holmberg, A.Casal and A.Baader: Force Strategies for Cooperative Tasks in Multiple Mobile Manipulation Systems. *International Symposium of Robotics Research*, Munich, October 1995.

[10] J. Košecká: *A Framework for Modeling and Verifying Visually Guided Agents: Design, Analysis and Experiments*. Dissertation, Grasp Lab, March 1996.

[11] M. E. Lesk and E. Schmidt: *Lex - A Lexical Analyzer Generator*. Computing Science Technical Report No. 39, Bell Laboratories, Murray Hill, New Jersey, 1975.

[12] J. Lygeros, C. Tomlin, and S. Sastry: Controllers for Reachability Specifications for Hybrid Systems. *Automatica*, 1999. Accepted for publication, to appear in March 1999.

[13] R.M. Murray, Z. Li and S. Sastry: *A Mathematical Introduction to Robotic Manipulation*. CRC Press, 1994.

[14] *QNX Operating System: System Architecture*. Users Guide, QNX Software Systems Ltd. 1997.