A Multi-agent Approach to Distributed Control for Task-level Programs for Cooperating Manipulators

Petra Bohner

Institute for Real-Time Computer Systems and Robotics
Faculty for Informatics, University of Karlsruhe
Kaiserstr. 12, 76128 Karlsruhe, Germany
e-mail: bohner@ira.uka.de
tel.: ++49 721 608 3545,
fax: ++49 721 60 67 40

Abstract

The use of two or more robots in a common workspace is essential to expand the field of potential applications. This paper presents a distributed approach for executing task-level programs for cooperating manipulators or one arm systems with many degrees of freedom. At task level, only the movement of the objects to be manipulated is specified. The according manipulator motions are then calculated by assigning local intelligence to each joint. The local intelligence enables the joint agents to calculate in parallel their appropriate movements to make the end effector reach the desired position. During execution, each joint evaluates sensor data to compensate for execution errors, to react on unexpected obstacles and to provide the manipulator coordination. Therefore, a distributed fuzzy rule base has been developed. As this local evaluation may lead to non-optimal overall behavior of the manipulator, the joint agents communicate to provide global suboptimality.

Keywords: theory of dependent, distributed agents, reactive and plan-based agents, distributed control, application of Distributed Artificial Intelligence

1. Introduction

The study of several cooperating robots or systems with many degrees of freedom is a topic of ongoing research. Potential applications range from manipulation in obstacle cluttered environments or narrow environments to medical applications, where many degrees of freedom are needed to ensure that the patient isn't hurt. A general concept for planning and controlling these systems even gains in importance with the recent developments of application specific manipulators [Ni92].

Work in the area of systems with many degrees of freedom concentrates on mobile robots ([AO92], [SF93], [SFu93]). However, approaches for mobile robot systems usually exploit some of the following properties which are specific to mobile robot systems: the single agents are decoupled, often of relative simple shape, sometimes even intersection regions can be identified which are the only possible collision regions for the system. Thus, the application of these approaches to manipulator systems isn't straightforward. Therefore, most existing systems for manipulators use global methods which can be classified as follows: a global task planner calculates some kind of elementary subgoals, a centralized global path planner generates the joint angles sequences to be executed, a control unit surveys the correct execution of these sequences ([TC93], [HL+91], [LZ87], [Le90]). There are, however, a number of shortcomings associated with the above systems that limit their usefulness:

- global planning is very time-consuming
- the systems are limited to a fixed number of robots
- the systems’ fault tolerance is limited to a small range of errors

Local methods based on the technique of potential fields are investigated by [Ba90] and [Mc93]. However, the integration of sensor data into their approaches isn't straightforward. A distributed approach to solve the inverse kinematic problem is presented by [Du93], but limited to the 2D-case for 3 redundant joints.

[Ka*93] provide procedures for error detection, but as they use a global path planner, these errors can only be corrected by global replanning.

Two approaches of sensor-based manipulator planning can be found in the literature: [Co89] proposes a distributed, behavior-based controller for a two degree of freedom manipulator being mounted on a mobile robot. The architecture is an enhanced version of the subsumption architecture proposed by Brooks [Br86]. Thus, the arm can grasp empty cans by evaluating sensor data as well as camera images. However, no global knowledge is integrated on any level of this system and no global optimality can be achieved.

Another sensor-based approach can be found by Lumelsky and Cheung [Che90], [Lu91]. They developed a control algorithm for a whole-sensitive arm manipulator whose whole body is covered with a sensitive skin to detect nearby objects. Their method is based on the calculation of the C-space and thus suitable for the manipulator with 3 degrees of freedom they
use, an extension to manipulators with more degrees of freedom is likely to be far too time-consuming. In conclusion, it can be stated that none of the above described systems seems suitable to meet the demands of the problem of generating motion commands from task specifications given as relative object movements for general manipulator systems with many degrees of freedom and then reliably executing the calculated movements by evaluating sensor data. In this paper an approach is described that allows to treat cooperating manipulators or one arm systems with many degrees of freedom in an equal manner: the task to be fulfilled is specified in terms of relative object positions. Each joint is regarded as an autonomous agent with local intelligence which enables it to plan and execute in parallel given tasks. The joint agents communicate to prevent themselves from falling into local minima due to the "relative shortsightness" of a single agent. Passing from one arm systems to cooperating manipulators just adds more degrees of freedom to the overall system as the double amount of agents negotiate the object position and the relative position of the manipulated object can be chosen in a free manner. Therefore, in the system described here most of the autonomy is given to the very fast execution level. A global planning level generates "hints" of the paths of the objects to be manipulated, a distributed fuzzy controller generates and executes the appropriate joint movements. Thus, tasks can be realized very fast, the system can react fast on environment changes and incorrect data through evaluation of sensor data. In addition it is fault tolerant, as defect motor drivers and controllers are replaced automatically; no reconfiguration or replanning is necessary. The paper is organized as follows: in Section 2, the overall system architecture is introduced. In Section 3, the on-line level is discussed for one manipulator. Then, the distributed position calculation, one of the basic properties of the distributed concept, is explained in detail. After that, the coupling of several manipulators is presented. In Section 4, examples are given which demonstrate distributed path calculation for one arm. Section 5 concludes this paper with suggestions for further enhancement of the architecture.

2. System Architecture

The general assumption made in this approach is that a manipulation task can be solved by planning and executing a sequence of object goals for the objects to be manipulated. These goals have to be realized simultaneously or in chronological order. Object goals are positions of objects (given by coordinates), trajectories of objects, forces to be exerted by objects and relative movements of objects (e.g. to rotate, to approach). The signification of the term "object goals" can be explained by the following example: to screw a bolt into a nut, the following goals have to be planned: the axes of bolt and nut have to be aligned, the objects have to be brought together until they touch. Then they have to be turned. Similarly, a part of executing a task can be to exert certain forces or torques on objects. The above described examples stresses the fact, that the use of cooperating manipulators add degrees of freedom to the systems not only due to the higher number of robot joints. When one manipulator positions a peg into a hole, the hole position is fixed. The cooperating manipulators however can fix the manipulation position with respect to optimality criteria to be negotiated. The overall system architecture is illustrated in Figure 1.

![Fig. 1: The overall system architecture](image-url)
has to be applied to compensate for friction) this force is generated on-line.

The off-line planning level can be reactivated during the execution phase if the on-line level fails when executing an object goal. However, most of the correcting and instantiation actions happen in the execution module. Thus the very time-consuming communication between the different planning levels which is substantial to most systems is eliminated in the approach described here. Another advantage is that corrections are executed on a low level that is running on high sample rates resulting in very quick reactions to problems.

Starting from descriptions of the task to be executed, the object goals are calculated in the object-level planning module. The object planning level plans as general as possible, as in many situations there are many possibilities to solve a given task. For example, to screw a bolt into a nut only the relative movement of the two objects is relevant. Therefore the goal can be achieved by executing the movement either along a horizontal or vertical axle. In our approach, this isn't specified during the task-level phase but is part of the distributed target intelligence level. Thus, the task-level planning is kept independent from environment and special manipulator features. The specialization happens in the distributed target intelligence level, where a fast and very flexible execution can be guaranteed. In our system, this flexibility is further enhanced by the planning architecture. It is thus possible to instantiate the relative movements accordingly or to change parameters in narrow environments or in environments cluttered with obstacles, so that the task can be fulfilled in any circumstance. If the relative movement is impossible to be fulfilled, replanning is necessary. In the system described in this paper it is assumed that the goals calculated on this level are treated as constants in the level beneath.

In the grasp planning level, the grasp positions are calculated. Therefore, physical properties (center of gravity) as well as heuristics (don't touch the thread if you want to screw) are evaluated. Details of the off-line planning level will not be regarded further in this paper.

The interface module takes the series of object goals and grasp positions as inputs and feeds them one by one to the distributed target intelligence module. Furthermore, it communicates with the object-level planning module to report success of the on-line execution or to initiate replanning. Later, this module will give on-line help to the distributed target intelligence module, e.g. if the manipulator gets stuck and will provide global knowledge. The distributed target intelligence module provides the motor commands for the joints' motor drivers. Therefore, each joint is regarded as an autonomous agent and its movements are planned and controlled by an individual fuzzy controller. (see section 3)

The fuzzy controller enables the joints to execute the object goals as well as to react on unexpected obstacles or inexact descriptions of the environment by evaluating the gripper sensor data or sensors positionned on the joints or links; the actual positions goals are backpropagated to the interface.

In our system, this distributed fuzzy control architecture is realized on a suitable hardware platform which consists of hardware modules representing the joint agents and other abstract modules. (e.g. the goal interpreter, see Section 3).

3. Distributed control and execution

Many existing approaches consider the manipulator as a whole during the path planning, execution and control phase. This causes several drawbacks:

- The path planning problem is NP-hard; existing planners with many degrees of freedom (>3) are extremely slow or use local approaches like the trial-and-error approach described in [Bo94]. Additionally, dynamic properties of the manipulator are very difficult to incorporate in this systems, and it is impossible to react in real-time to unexpected events.
- The systems are not fault-tolerant and break completely down in case of the failure of one joint or the controlling processor.

This approach works very different. Only the trajectories of the objects to be manipulated is preplanned on a high level. Trajectory preplanning for the joint motions isn't done at all but the joint motions are calculated on-line using a distributed fuzzy rule base. The on-line system architecture for one manipulator is shown in Figure 2.

3.1 Concept of distributed target intelligence

The heart of this on-line system is the distributed target intelligence (DTI) module which will be explained in detail in this section.

Inputs in the DTI are the object goals calculated off-line. These can be the positions of the objects, trajectory types of the objects (straight lines, parabola...) forces or torques to be
exerted on objects. Secondly, sensor values from external as well as internal sensors are considered as inputs in this unit. Each joint is regarded as an autonomous agent with its own fuzzy rule base. A goal interpreter evaluates external sensor data and propagates the goals to joint agent1. Joint agent1, which is the agent closest to the object then tries to reach the object goal by evaluating its fuzzy rule base. The generated movement is executed, the results propagated to the next joint agent, which continues appropriately. After each joint had the chance to react, an inverse coordinate transformator then calculates the new object position and communicates it back to the goal interpreter. The interpreter interprets the current state and propagates it to the joint agents again. This control loop keeps the actuator moving until the current goal is considered to be finished. The next goal is then considered. The object positions are communicated to the interface module as well. In case of successful termination of an object goal, the interface provides the next goal. The interface reports successful termination of a task to the object level planner. When the interface module detects a dead-lock, it communicates this to the goal interpreter to stop execution and to the object-level planning module which starts replanning.

In each fuzzy rule base, three properties are encoded:

- sensor values are evaluated; deviations are corrected and unexpected obstacles are avoided. Fuzzy rules treating sensor evaluation are prioritized.
- each joint tries to realize the object positions and forces. As this leads to a minimization of the distance to the position and force to be reached it can be proved that the goal will be reached. The remaining differences are propagated to the next joint agent. The fuzzy realization of position calculation will be described in detail in section 3.2.
- solutions to redundant kinematic problems will be implicitly calculated on this level.
- the dynamic properties of the joint are realized, e.g. acceleration and deceleration possibilities.
- in case of defect units (crash, overheated motors,..) the functionality of the concerned unit is automatically provided by another unit.

3.2 Fuzzy realization of object position

As a testbed for the distributed object position calculation a PUMA-like six degree of freedom manipulator has been chosen. As the joint agents have to calculate and move in parallel, independent appropriate features had to be developed. They were chosen as follows: For each joint, the goal position as well as the actual hand position are projected on the plane normal to the rotation axes and passing through the origine Oi of the according reference frame, see Figure 3 for joint 1 and the first three joints.

![Figure 3: Calculation of movement criteria for joint 1](image-url)

The angle between OPg and OPe then serves as a criteria to be minimized (these angles equal zero when the end effector has reached its goal position). Simulation results proved that any start/goal combination can thus be reliably calculated. Fuzzy rules are used to control the velocities (accelerations) of the different joints. Therefore, the relevant angle difference ANGLE_I and the difference of the actual velocity to the maximal velocity as well as the actual acceleration are taken into account. The resulting rules are like

\[
\text{IF ANGLE}_1 \text{ IS BIG THEN ACC = BIG}
\]
\[
\text{IF DIF VELOCITY IS HIGH THEN ACC = BIG}
\]
\[
\text{IF ACC IS HIGH THEN ACC IS NOT NEG HIGH}
\]

These provides regular motion of the single joints. For an example of an executed path, see section 4.

3.3 Manipulator coupling

In the case of several manipulators, the DTIs are coupled with their goal interpreters, see Figure 4.
The goal interpreters communicate for initialization and adaptation of the relative positions and movements and in case of closely coupled object goals. A suitable sensor architecture which allows the manipulators to determine their relative positions supports this communication as it is possible to avoid the cummulation of sensor errors that is typical for the long chain of angle sensor signals that are used to determine the absolute actuator position.

The whole concept of on-line planning and sensor evaluation demands fast reactions to guarantee that the system can react in real-time. The hardware platform used to support this is the PEAR architecture that is described in [Ma94]. The PEAR architecture is especially designed to suit fast and complex control applications. The concept is to set up a control computer that is perfectly tailored to the application. to achieve this two levels of modularity are realised. the first level is the functionality of a processing element. Each processing element can be expanded individually, like in a VME-Bus system.

The second level of modularity is the interconnection of processing elements to functional modules. The basic idea of the PEAR architecture is to set up the communicational structure accordingly to the application. this approach exploits the fact that control applications usually have a very static structure. As the computer can be tailored to this structure it is possible to implement only the communication lines needed. This restriction of the number of communication lines enables extremely high communicational performance. The performance is achieved by placing the processing elements physically close to each other. In this way the theoretical maximum of communication speed that is defined by the speed of the processor is made available. As communication is as fast as a memory access it isn't the bottleneck and the limit for the maximum number of processors any more.

To implement the DTI one processor will be used for each joint agent, this is done to profit from the following advantages. If the system is to be adapted to a different actuator the number of joints can easily be changed by simply adding joint agent processors. If one of the joints is of a completely different type (for example a linear motor) just the software and motor driver hardware of one processor needs to be changed leaving the others untouched. The processor responsible for a joint can easily be specialised on it's job by being expanded with joint specific position or status sensors (for example motor temperature, motor load or collision sensors). As the algorithms for different joint agents are executed on different processors it is guaranteed that they don't influence each others timing or data except by passing information over the clearly defined interface. The high degree of parallelisation results in very high sample rates and therefore in very quick reactions.

As the PEAR architecture enables application specific expansion of the processors the joint agent processors will be equipped with an additional fuzzy logic co-processor to speed up evaluation of the fuzzy rules. Additionally there will be interface boards to the sensors and motor drivers. The cartesian coordinate transformation unit will be specialised on its job by being upgraded with mathematical co-processors while the sensor processors assisting the goal interpreter will need sensor controllers and additional analog input boards.

The mechanical principle of the PEAR architecture realises these processing elements as stacks of printed circuit boards. These processing elements are placed on a carrier board resulting in a parallel computer specifically set up for the actuator.

The computer structure is expected to be able to run the algorithms with sample rates of 10kHz and more. This enables very fast reactions to sensors as well as the control of very fast robots.

4. Example

Figure 5-7 show an example of a calculated path. In order to enhance the clearness, only the two-dimensional case is shown. When given a cartesian start - and goal position, the planner calculates a serie of about 40 joint angles to pass from start to goal.
5. Conclusion

A fast distributed architecture for executing task-level programs for cooperating manipulators has been presented in this paper. The system realizes planning and execution of manipulation tasks in real time. Beyond that, the architecture provides fault tolerance by distribution of intelligence and reactive behavior through on-line sensor interpretation. The distributed planning and execution architecture proved to provide good results in path planning in two and three dimensions. Actually, optimization concepts and concept to provide sensor based collision avoidance are being integrated in the system.

In this approach, a suitable generation of intermediate object positions guarantees the successful on-line execution. An extension that is currently being examined concerns the evaluation of the distributed target intelligence by an intermediate level which has global knowledge concerning environment and object geometry. This intermediate level could further enhance this architecture. Then in case the manipulation gets stuck, certain movements could be undone so that local execution can be continued without contacting an off-line planning level.
Acknowledgement

This research was performed at the Institute for Real-Time Computer Control Systems and Robotics, Prof. Dr.-Ing. U. Rembold, Prof. Dr.-Ing. R. Dillmann, at the University of Karlsruhe, Germany. It was sponsored by the Commission of the European Communities (ESPRIT-BRA No. 6769 SECOND). Thanks are given to Prof. Dr.-Ing. Rembold and Dr.-Ing. Raczkowsky for many discussions on earlier drafts of this paper. Special thanks are given to B. Magnussen for many suggestions and contributions for the work described here.

Literature


[Ma94] Björn Magnussen. A Parallel Control Computer Structure for Complex High Speed Applications. Accepted for the First Asian Control Conference, ASCC, July 1994, Tokyo, Japan


