TASK-LEVEL PROGRAMMING WITH COLLISION AVOIDANCE FOR AUTONOMOUS SPACE ROBOTS

P. Adolphs and J. Matthiesen

TH Darmstadt, Control Systems Theory and Robotics Laboratory, Schlossgraben 1, D-6100 Darmstadt, Germany

Abstract. The reasons for integrating collision avoiding path planning into a task-level programmable multi-sensor robot system are put forward. The underlying system architecture and the specific approaches for environment modelling, task planning and path planning are discussed. Task planning is performed using a rule based expert system and a frame representation of relevant environment data. Path planning is based on a configuration-space approach with a fast new algorithm for obstacle transformation. Results gained from experimental laboratory work are presented and show some advantages and problems of the entire system.

Keywords. robots; task-level programming; collision avoidance.

INTRODUCTION

In this paper two major problems of orbital robot system autonomy, facilitated task-level programming and real-time collision avoiding path planning are discussed and an integrated solution is put forward. Task-level specification of system behaviour is a key issue serving the reduction of data transfer bandwidth requirements. The underlying idea is that when manipulating within the space environment a ground or astronaut operator merely has to state the system's handling objectives in a more or less abstract fashion leaving the planning of details and the entire execution train to the control system. One necessary feature of a partly autonomous action planner is the ability to generate collision-free trajectories for the manipulator including the consideration of real-time varying environments. Extensive theoretical and experimental lab work has shown collision avoiding path planning in a cluttered environment to be a considerable problem of its own. The existence of a real-time path planner therefore greatly alleviates the burden the general robot control system's strategic and planning level has to carry. Alternative approaches suggest disjunct use of the workspace by separate manipulators (Cheng, 1991) and tackle the problem of single manipulator collision avoidance by explicit environment and task design. Comparable approaches (Lozano-Perez, 1989) have only limited real-time capabilities or have so far not been demonstrated in an integrated system featuring task-level programming.

SYSTEM DESIGN

The functionality of the entire system has been distributed on a server and a client level named in analogy to the X-windows terminology (Nye 1989) for modularity reasons. Major benefit of this subdivision scheme is that improvements can be added on either side of the layer interface without interfering with the other side's integrity. The task-level programming features belong to the strategy and planning (S&P) "client" level and have been tested with an existing system for knowledge based assembly shortly sketched in the following section. Collision avoidance is considered a feature of general use and was consequently integrated into the action execution "server" level.

Task-level programming

The aim of using a higher language task specification of programming language is to facilitate the man-machine communication by adapting the machine's level of data processing in some sort to the common way humans think. Computers need formal languages to process and can only work on a restricted vocabulary. Humans on the contrary like to express their intentions individually and can handle multiple modes of expression and even syntactically wrong sentences. For the purpose of high level commanding a multi-sensor robot system we defined a formal language frame permit-

ting the usage of near to natural language constructs with a simple syntax and individual wording. We call the task-level commands "operations" whereas the more primitive actor motions or sensor functions are referred to as "(elementary) actions". An "operation" consists of three elements answering the following questions:
a) What is to be done? - the **operator**, b) Which tool shall be used? - the **gripped object** and c) Where is the focus of attention for this operation? - the **target object**. Thus, an operation reads e.g. "put, probe_1, freezer". A series of operations is put into a list called the Assembly Plan or Operation Plan. This list is given to the S&P modules for processing. These modules have to bridge the gap between the action and the command level.
The action level integrates a number of services provided for the S&P layer such as elementary robot and gripper motions configurable with parallel sensor (eg. force-) control and observation modes. When a motion finishes, a batch of current sensor data (e.g. wrist forces/torques, robot & gripper positions, grip force and object presence/slip information) are gathered and preprocessed. Together with action state information (e.g. that a motion was stopped due to contact detection) this batch is transferred back to the S&P level.
The coarse task-level operations are split up by the S&P modules into a series of subordinate actions according to the initial state and the intermediate sensor data readings. This subdivision answers to a) Which component (actor/sensor) will carry out the following action?, b) Which specific action type is necessary? and c) Which are the appropriate parameters for the action? Following the action execution its result must be classified based on the sensor readings and execution state information. If a deviation from the intended course of execution is detected, corrective measures have to be taken. This means, the selection of the next action is based on the intermediate state generated.
The problems involved include logical planning and classification. We used expert system technology to further structure the S&P level. One key aspect in conjunction with expert systems is the separation of knowledge from the knowledge processor or inference engine. The knowledge is subdivided twofold into factual vs. strategic on the one hand and static vs. dynamic or time-varying on the other. Static factual knowledge subsumes the naming, fixed characteristics and hierarchical structure of the environment objects. Dynamic facts comprise position information, system generated symbolic states and information about mechanical and logical object interconnections subject to change due to robot handling. Strategic knowledge is represented as production rules of the form

IF (the robot is in approach position of probe_1)
  and (the gripper is closed)
THEN (open the gripper)

The set of conditions and actions implemented for system control, planning and classification adapts the application independent forward chaining rule processing mechanism to the problem domain.

Implementation

The task-level robot programming system WISMO developed at TH Darmstadt (Simon, 1991) features frame-based hierarchical object modelling and a production rule based cyclic planning and execution control algorithm. Rules are used in every step of this cycle: manipulator/sensor action type selection, parameter calculation, system state classification after action execution and environment model updates. The system thus emulates a planning-acting-perception cycle. The WISMO architecture is shown in Fig. 1.

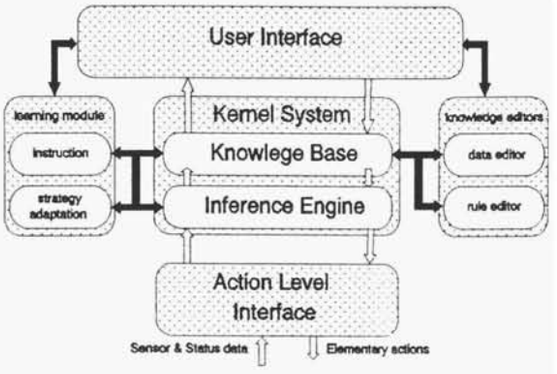

Fig. 1: WISMO architecture (Simon 1991)

A WISMO environment object is represented as a collection of positions relative to a reference coordinate frame in work space together with appropriate handling information. The positions are of predefined type (eg. denoting reference to world or object coordinates, defining tool corrections or part-of relations) in order to enable automatic algorithmic handling. The manipulation states of robot, gripper and objects are expressed using a predefined set of symbolic attribute-value pairs that the system classification component is able to deal with. Data reduction was achieved by superimposing a class hierarchy and an inheritance scheme for positions and attributes.
Knowledge transfer to WISMO is performed in three steps: First, a rough outline of the course of action is laid down as a list of operations, e.g. "mate, peg_1, hole_3". In a second step, the objects involved have to be modelled and their relevant position and attribute data have to be obtained. Thirdly, the operator interactively specifies which

concrete manipulator actions he wishes the system to perform given the operations. A recording module memorizes the initial symbolic system state and the actions/parameters commanded and transforms both into action and parameter planning rules which consequently enable the system to reperform the planning shown by the human instructor. Another positive aspect lies in the instruction mode's basic strategy to fasten the strategies told only to the classes of the objects involved. By this means basic substrategies like peg-in-hole mating or screwing must be instructed only once and are automatically remembered by the system given a similar problem. Using this simple machine learning approach the tedious process of manually filling the system's rule base is reduced.

Collision avoidance

A very important feature of a task-level programmable robot sytem is autonomous collision-free path planning. Such a component frees the task planner from geometrical considerations concerning the robot's movement: Provided that such a component exists, it is sufficient to command target configurations with respect to the given task, for example the position of an object which should be grasped. The path-planner computes automatically a movement which guides the system to the target position.

However, in order to render the problem handable, the following classification of the overall manipulator movements is advantageous: For objects which have to be handled, we define a pre-position from which the grip configuration can be approached collision-free under all circumstances (Kegel, 1990). Then, the manipulator movements decompose into three parts: gross motions between pre-positions, fine motions between pre-positions and grip positions and, finally, the gripping act itself.

In order to ensure the flexibility of such a robot system especially in dynamically varying environments the path planner has to work online. Therefore on the one hand the internal model of the environment has to be updated whenever a change occurs and on the other hand the computation of the further movement must be very fast. To attain this goal an internal model of the environment is described in configuration-space (c-space), which is the space of all kinematically possible configurations of the robot. All configurations which cannot be reached without collision between robot and an environment obstacle are marked as forbidden. Main advantage of the c-space model is a fast collision-test for the robot which can be performed by a single read access to the c-space. Due to this fact a very efficient path-planning is possible. Unfortunately, the transformation of objects given in work-space coordinates into c-space needs much computational expense. In order to reduce this time considerably some special algorithms were developed, which are described im more detail in the next section.

Representation of environment

Mapping objects given in work-space coordinates into c-space is a very time-consuming task, especially when considering the complex shape of the robots links not only by simple envelope-bodies. Most approaches use recursive algorithms (Gouzenes, 1984), which start at the first link of the robot and determine the collision interval when moving the link. To avoid this on-line collision check we developed a new method. Therein the correspondence between work-space and c-space is stored in a look-up-table, which is called OCMEM-table.
This concept is based on a discretization of work-space as well as c-space into elementary cells. Figure 2 shows all the positions $V_a(e)$, which can not be reached by the robots tool-center point (TCP) without a collision between the robot and such an elementary cell e. Thereby the orientation of the robots hand and a possible load is fixed and the TCP is defined as the intersectionpoint of the three hand-axes. Thus a TCP-position complemented by the kinematic state corresponds to a specific shape of the robot. To store the correspondence between e and $V_a(e)$ in a look-uptable is the basic idea of OCMEM. Because the table entry does not depend on the actual environment but only on the size of the elementary cells, the table can be calculated with an off-line procedure.

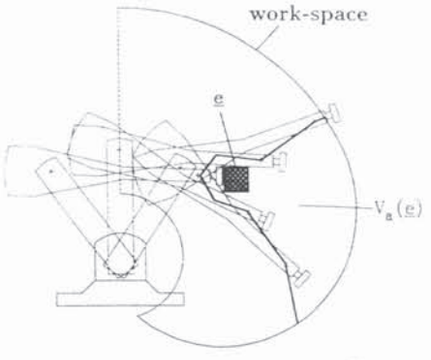

Fig. 2: The elementary work-space cell e causes the not collision-free reachable region $V_a(e)$ (=OCMEM lookup-table) in c-space.

For mapping an object O in real-time to c-space in the first instance it has to be discretized into elementary cells like

$$O = \{ e_1, e_2, \ldots, e_m \}$$

In the next step the OCMEM table is applied to every cell e_i and the corresponding forbidden regions in c-space are superposed. Thus the not collision-free reachable region C(O) with respect to the object O is calculated via

$$C(O) = V_a(e_1) \cup V_a(e_2) \cup ... \cup V_a(e_m).$$

The principle of this transformation is illustrated in fig. 3. The object O is discretized into 3 elementary cells (a). The OCMEM table contains for each elementary cell e_i the non-reachable c-space regions $V_a(e_i)$ (b), which are superposed to get the whole forbidden part of the c-space C(O).Because

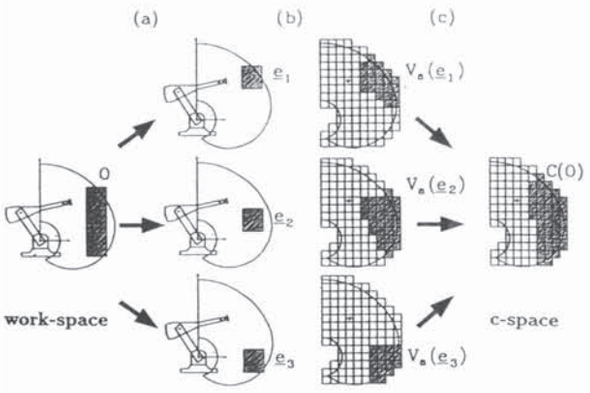

Fig. 3: Mapping the object O to c-space. (a) Discretization of object O. (b) Determination of the forbidden c-space regions for each elementary-cell. (c) Superposition of these regions.

of rotational symmetry with respect to the z-axis the TCP-position is described using cylindrical coordinates (r,φ,z). With a dimension of the elementary cell e of 5cm x 5cm x 2° the size of the OCMEM table would be in the range of 100 Megabytes, which today cannot be realized on standard hardware. To make this concept applicable to an industrial robot system some special techniques have to be used in order to reduce the size of the OCMEM table as well as to speed up the mapping. Therefore the symmetry of the workspace according to the z-axis is used, which results in a considerable reduction of size. For the example discussed above we achieve a reduction factor of 180. By additionally using a difference-coding technique which is described in (Adolphs, 1990) a very short mapping time is achieved.
By using this approach all objects, whose position is known before the robot movement starts, can be mapped to the c-space. This task can be performed in a few seconds. But moreover changes in the environment which occur while the robot is moving have to be considered for the internal model. For this purpose an extension of the OCMEM-concept described so far for moving obstacles was developed which cannot be discussed here. A description is given in (Adolphs, 1992).

Path-planning

The shortly discussed method for modelling obstacles into the c-space is well suited to apply many of the well-known global path-planning methods. To achieve fast computation we developed a special path-planning algorithm which is based on the distance-field approach, but includes some heuristical motivated enhancement.

Low computational expense is reached by utilization of special limitations. These limitations are:

1. Restriction to two directions for searching the path
   (a) random search within a r,z-slice of c-space
   (b) search in φ-direction

2. Priority on finding one possible path, not necessarily the optimal one.

This results in a very fast algorithm which is able to find semi-optimal paths. It is specially appropriate to meet the online requirement.
Fig. 4 illustrates the principles of this MINPOT algorithm in more detail. The figure shows a 2-dimensional z,φ-slice with constant r-coordinate in the c-space described in cylindrical coordinates. This simplification is made for better illustration only, all the algorithms are implemented for three dimensions (including the r-coordinate). The hatched area denotes the non-reachable regions in c-space. Starting at the target cell (T) a distance field is generated within the target sector. The cells with the same φ-coordinate as the target cell (target sector) get a distance value, which describes the distance to the target cell. Thus the neighbouring cells get the value 1, the neighbours of these cells the value 2 and so on. The generation of the distance field in that manner is related to the type (a) search discussed above.

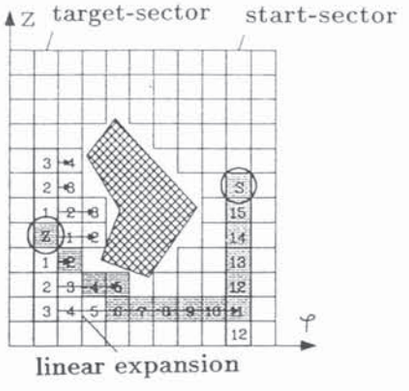

Fig. 4: Basic idea of MINPOT-algorithm.

Every cell that gets a distance value while expanding the field serves as a starting cell for a linear

expansion in φ-direction. For the example of fig. 4 the linear expansion is successful in the third step of expansion, because the start-sector is reached first in this case. Now the generation of the distance field continues beginning with the cell in the start-sector which was reached first.
For more complex environments the algorithm has special strategies for the case that the startsector cannot be reached in the first step: Additional sectors for expansion of the distance field are located between start- und target-sector automatically.
The path can be found by following the negative gradient of the distance field starting with the start-cell. The path found is highlighted in fig. 4 by the grey area.

Sensor-systems

Besides the fast algorithms for modelling and path-planning, sensor systems are needed which are able to survey the environment in real time. To limit the hardware expense of such systems we use a hierarchical classification of the objects: objects whose position in the environment is fixed are modelled in advance. Thus the sensor systems can concentrate on changes within the work-cell.
As an example for demonstrating the real-time capabilities of the path-planning module an infrared-radar sensor was integrated, which surveys a table. When an additional obstacle is put on the table this sensor determines the current position of the object and sends the geometrical data of this obstacle to the path-planning processor. The robot avoids such obstacles automatically without interrupting its motion.

Integration

One main aspect of the work presented in this paper is the integration of task-planner and path planner in a modular manner. Fig. 5 shows how this integration was realized. The task planner generates commands which are classified in the following way:

(1) motion commands with collision avoidance for gross motions
(2) motion commands without collision avoidance for fine motions
(3) gripper and/or manipulation commands

Gross motions (1) which implicate long sweeps to a target configuration are given to the path planner first. It generates a sequence of intermediate points which describe the collision free path. Commands of type (2) are transferred directly to the RCM 3D commercial robot control unit. Commands belonging to the third type are sent to and are autonomously executed by the gripper control unit.
By means of integrating sensors into the system the environment can be monitored online: information about the objects handled is generated by the gripper sensors and fed back to the task planner. On the other hand, sensors surveying the work space directly provide the path planner with data for updating its internal world model.

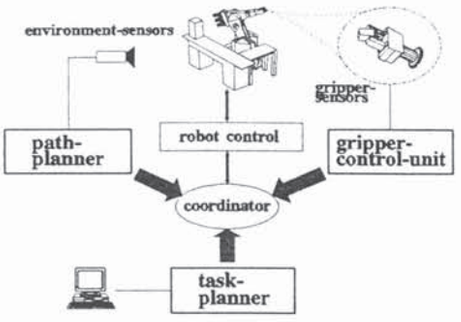

Fig. 5: System hardware structure

Experiments

The entire system has been verified in ground-based experiments and continues to be used. In the course of system build-up a MANUTEC R3 robot connected to a Siemens RCM-3D control unit was augmented by the components discussed (cf. Fig.5). For the sake of modularity task planner, path planner, gripper-control unit and coordinating unit are implemented on separate hardware. This permits modifying the setup with respect to the specific task. For instance, the adaptation to different grippers (we used a parallel jaw gripper and a three-fingered gripper as well) or different sensors for surveying the environment is possible.

Task specification

The modelling and instruction scheme proposed turned out to be effective in a series of lab setups including the handling of a satellite mock-up (Matthiesen 92) and several small assemblies. The time for implementing an assembly consisting of 60 operations, 12 objects and including strategies such as tool handling and screwing was about two weeks. Students unfamiliar with the robot environment and the expert system approach were able to use the system after about a week of training.
Experience showed, however, that by instruction in general too many simple rules are generated. This results from the underlying assumption that every planning step has its own specific reason which in fact turns out to be rarely true. Debugging large rule domains is time consuming and requires a

degree of expertise the target system user is not expected to own. Taking into account, in addition, that planning rules represent links of a decision graph we will further investigate into using a graph representation for planning.
Some difficulty also arose from the approach not to bother the instructor with the situation specific interpretation of sensor data. One way to cope with this requirement we went is to use standard attributes for classifying the situation such as the symbolic robot position and gripper state (open, closed, object gripped etc.). However, the evaluation of any action's success is tricky on such premises and further work has to be invested in this area.
Finally, the geometrical modelling should be supported by CAD techniques and in the same step more geometrical object information should be supplied in order to enable more complex sensor data interpretation and knowledge based updates of the collision avoiding path planner's object database.

Path-planner

The algorithms for modelling the environment and path-planning were implemented on a INTEL i860 processor running with 32 MHz. Using this processor the calculation of internal world model as well as path-planning is accelerated by a factor of 10-15 with respect to a former implementation on 80386 (16 MHz) hardware.
A reasonable compromise between low computation time and accuracy could be achieved by a size of elementary cells of 5cm x 5cm x 2°, which was evaluated with a lot of simulations. The mapping of an additional obstacle into the c-space model need 50-100 msec. depending on the size and the position of the object. A global path-planning with respect to whole model of environment can be performed in 15-100 msec.. Thereby the computation time highly depends on the length of the path which has to be computed. Fortunately the time for path-planning does not increase for more complex environments. It depends on the path-length and the size of elementary cells only.

CONCLUSION

An autonomous multi-sensor robot system was discussed which can be programmed using task-level instructions. In this context the capability to automatically generate collision-free robot trajectories complemented with suitable environment obstacle sensing and local dexterity by using flexible multi-sensor grippers plays an important role.

ACKNOWLEDGEMENT

This work was partly supported by DFG (collision avoidance) and partly performed in a project sponsored by DARA and headed by ISRA Systemtechnik GmbH. The authors are grateful for this support.

REFERENCES

Adolphs, P., D. Nafziger (1990). A method for fast computation of collision-free robot movements in configuration-space. IEEE Int. Workshop on Intell. Robots and Systems, Tokyo.

Adolphs, P., H. Tolle (1992). Collision-free real-time path-planning in time varying environment. IEEE Int. Conf. on Intell. Robots and Systems, Raleigh, USA.

Gouzenes, L. (1984). Strategies for solving collision-free trajectories problems for mobile and manipulator robots. The Int. Journal of Robotics Research, Vol.3, No. 4.

Cheng X., D. Kappey and J. Schloen (1991). Elements of an Advanced Robot Control System for Assembly Tasks. 5th int. Conf. on Advanced Robotics, Pisa.

Kegel, G. (1990). Erhöhung der Autonomie von Robotersystemen durch multisensorielle Informationen und Nutzung einer Wissensbasis. Dissertation, TH Darmstadt, Fachgebiet Regelsystemtheorie und Robotik.

Lozano-Perez T., J.L. Jones, E. Mazer and P.A. O'Donnel (1989). Task-Level Planning of Pick-and-Place Robot Motions, IEEE Computer, March 89, pp 21-29.

Nye, A. (1990). X Protocol Reference Manual for X Version 11. O'Reilly & Associates, Inc., Sebastopol, CA.

Simon W. (1991). Untersuchungen zu einer intelligenten und lernfähigen Robotersteuerung für die Montageautomatisierung. Dissertation, TH Darmstadt, Fachgebiet Regelsystemtheorie und Robotik, published in VDI-Series 20: "Rechnerunterstützte Verfahren", Nr. 47.

Matthiesen J., H. Tolle, S. Wienand, E. Ersü (1992). Sensorinformationsverarbeitung und Planung in einem autonomen Steuerungssystem für Wartungs- und Reparaturaufgaben in der Raumfahrt. Fachtagung Automatisierung, Dresden.