ELSEVIER

Automation in Construction 3 (1995) 305–320

AUTOMATION IN CONSTRUCTION

A behavioral language for motion planning in building construction [☆]

Rudi Stouffs [a], Ramesh Krishnamurti [a,*], Irving J. Oppenheim [b]

[a] *Department of Architecture, Carnegie Mellon University, Pittsburgh, PA 15213 USA*
[b] *Departments of Architecture and Civil Engineering, Carnegie Mellon University, Pittsburgh, PA 15213 USA*

Abstract

This paper reports on the development of a motion language to describe the behavior of automated agents in a dynamically changing environment. The language is based on a general representation that uniformly characterizes a wide variety of automated agents, using motion rules as a means of establishing the configuration of an agent at any given time. Motion planning is briefly discussed and the rudiments of a general three dimensional motion planning algorithm are presented. The results are illustrated in a simulation program with application to building construction.

Keywords: Building construction; Simulation; Obstacle avoidance; Path planning

1. Introduction

Building construction poses challenging problems in robotics for multiple reasons: site characteristics, man–machine cooperation and the diversity of automated agents. Of particular interest (to us) is this last issue, the use of a number of different and distinct agents in the construction of buildings.

Simulation of the construction process offers valuable insights into construction planning and management. The use of automated agents adds an extra dimension to this simulation. Apart from visualization of the construction process through consecutive projections of the building during construction, it provides a manifestation of the flow of material from storage to assembly and the manner of assembly itself. Moreover, the use of agents allows for a more precise analysis of the difficulty and time involved in the transportation of the material on the site. These inclusions necessitate a comprehensive representation for a set of diverse agents as well as general three dimensional spatial and motion planning algorithms to specify the requisite motional behavior of agents in a dynamically changing environment.

In this paper we elaborate on spatial path planning, describe a general representation for robot agents, introduce a motion language that allows one to capture the diversity in behavior

[☆] Discussion is open until August 1995 (please submit your discussion paper to the Editor on Construction Technologies, M.J. Skibniewski).

[*] Corresponding author.

between different robot types, and touch upon three dimensional motion planning. We illustrate the results in a demonstration of the RUBICON simulation program.

1.1. Automated agents

We define a *robot* to be any form or type of automated mechanical manipulator which takes part in the construction process and requires spatial motion of itself. Current research in robotics, whether in automation programming languages or path planning, is mostly focused on general-purpose manipulators such as arm-like manipulators. However, building construction offers a wide variety of tasks that often require task-specific robots. We can recognize potential use of assembly manipulators for transporting, handling or positioning construction materials, general-purpose interior robots for operations such as welding, grouting and nailing, and highly task-specific robots in areas such as concrete surface leveling or finishing.

As such, robots in building construction come in a large variety, each tailored to a specific task or class of tasks. At the same time robots may vary largely in size and, therefore, in mobility. Given the large size of the workplace and the fact that a task is rarely limited to a specific area within this workplace, issues of mobility and transportability play a far greater role in building construction tasks than in general manufacturing tasks. Often, the deployment of robots may additionally require cooperative human effort such as in final positioning.

In this paper we focus on manipulators that carry out pick-and-place assembly tasks. The results can be applied to other types of manipulators as well. The simulation examines robot motion from its initial location to the goal location; the specific actions involved with picking or placing are not simulated. Uncertainty is not considered. Thus, the simulation involves planning a collision-free path from a determinate initial configuration to a determinate goal configuration of the robot. Such motion is generally referred to as *gross motion*; it does not involve sensing or accuracy control.

1.2. Language for simulating behavior

A behavioral approach to robot motion for assembly programming has been introduced by Petropoulakis and Malcolm [1] (also [2], [3]). Their approach is to translate high-level assembly tasks into specific robot motions in order to deal with such problems as theoretical complexity and questionable functionality. Behavioral modules are designed that are predominantly concerned with the basic robot activities with respect to the objects of the assembly and their manipulation. These modules form the elementary units of the assembly system and are to be appropriately combined into higher level behaviors in order to perform complex part manipulations and assemblies.

Apart from concerns of translating high-level task plans into series of robot motions, the research reported in this paper is specifically geared towards a representation for different robot types and an expression of their behavior. As such, describing the behavior of a robot is not a means to an end, but the end altogether. We are particularly interested in finding a collision-free path for a robot from one location to another in the dynamically changing environment of a construction site. We are concerned with the part that a particular robot plays in the construction process, and with its place in that environment. The relationship between a robot and the environment is constrained by a number of factors: physical obstacles, interaction as specified by the tasks, interaction with human agents, safety considerations, and other issues that may be important to the robot's behavior.

In order to specify the appropriate behavior for a robot with respect to each of these constraints, without restricting the variety of robots that may be involved in the construction process, a motion language for robot behavior is developed. The language is used to specify the translation process of high-level pick-and-place tasks into specific, discrete, robot motions, taking into account motion constraints. The language is general in that it can describe disparate robot types, each operating in a multitude of different situations, yet it is focused towards a single purpose,

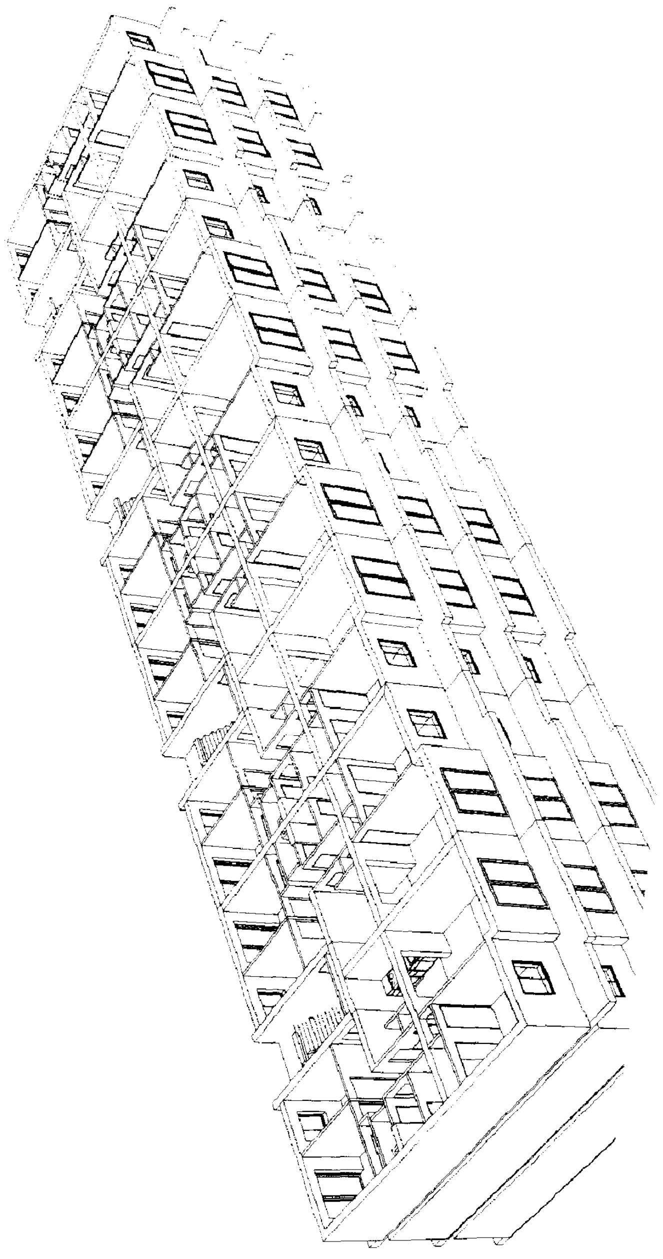

Fig. 1. Typical 40 unit building construction.

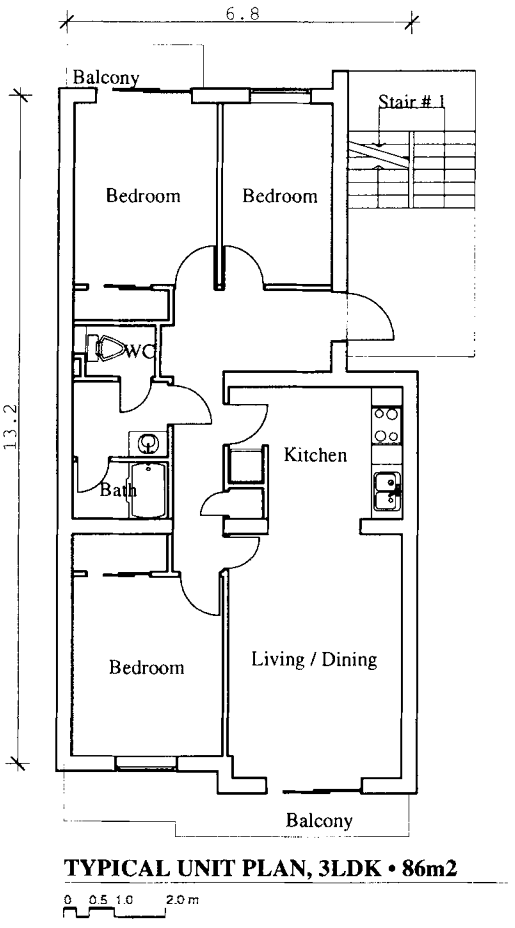

Fig. 2. Floor plan of a single unit.

that of specifying robot motions for use in a general motion planning algorithm.

1.3. Construction project example

To demonstrate the simulator we chose the construction of a typical Japanese precast concrete residential building. Figure 1 illustrates three stories of a typical 5 story, 40 unit building; Fig. 2 shows the floor plan of a single unit.

2. Representation

The representation of a robot must, before all, reflect the intended usage. Robot motion or path planning requires an expression of the robot's motional capabilities and of its swept volume. The former constitutes the basis of any motion planning in free space in the absence of obstacles; the latter allows for collision detection or the determination of free space. A representation of the motional capabilities of a robot relies in the first instance on an expression for the robot's configuration, where by *configuration* we mean both position and orientation of the body in question.

2.1. Configuration

A manipulator is typically an open kinematic chain of rigid links, connected with joints which allow relative motion of neighboring links [4]. At the "free" end of the chain we recognize the end-effector of the manipulator. Depending on the intended application of the robot, this end-effector may be a gripper, welding torch or other device. The base of the robot defines the other end of the open chain. This base may be either fixed in space or allowed to move within constraints established by the mobility mechanism. In this paper, we are not concerned with the physical properties of the chain of links and joints that relate the end-effector to the base, but only with the configurational relation. We need to know how the configuration of the end-effector is dependent on the configuration of the base, and which parameters control the relationship.

In order to describe the position and orientation of a body in space we rigidly attach a coordinate system or *frame* to the body. Such a frame can also be interpreted as a description of one coordinate system relative to another. We adopt the term transform to denote such a mapping between frames. A minimal description of a robot involves the following three frames: the universal frame $\{U\}$ that represents the outside world and is used as an absolute reference; the base frame $\{B\}$ that defines the configuration of the robot's base; the tool frame $\{T\}$ that defines the configuration of the robot's end-effector (see Fig. 3). The object frame $\{O\}$ defines the configuration of the material being handled.

The positional and orientational relation of these frames to one another is expressed by the

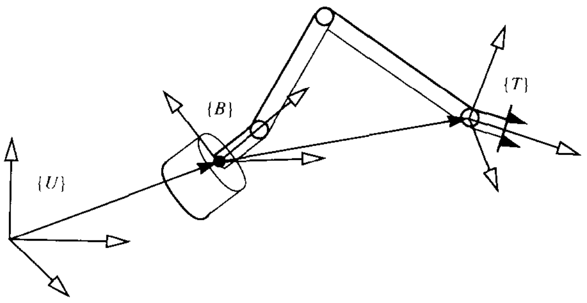

Fig. 3. Three frames to describe the configuration of a robot: the universal frame {U}, the base frame {B} and the end-effector frame {T}.

following two transforms: ${}^{U}_{B}T$ describes the base frame relative to the universal frame; ${}^{B}_{T}T$ defines the tool frame with respect to the base frame. The relation between any two of the three frames can be arithmetically expressed as a function of ${}^{U}_{B}T$ and ${}^{B}_{T}T$, e.g., the composite transform ${}^{U}_{T}T = {}^{U}_{B}T {}^{B}_{T}T$ which describes the tool frame relative to the universal frame. The object frame is described relative to the universal frame by the composite transform ${}^{U}_{O}T = {}^{U}_{T}T {}^{T}_{O}T$. A transform may be represented as a combination of a rotation matrix and a translation vector. The actual representation is unimportant as to the concept or usage and, therefore, will not be expanded on within this paper.

Even though the exact configuration of the robot requires a description of the orientation and/or position of each link, in most cases the robot is sufficiently described by the configuration of its base and end-effector. Robots in contemporary building assembly do not require a high degree of accuracy, because of the site characteristics and the size of construction elements involved in assembly, and because of man–robot cooperation. However, if a more precise description is necessary, it is straightforward to expand the number of frames involved in the representation.

2.2. Control parameters

Physically, the relationship between the end-effector and the base of the robot is defined by a set of links and joints, generally formed into an open kinematic chain. The number of degrees of freedom of the robot's end-effector, with respect to the base and for an open kinematic chain, corresponds in most cases to the number of joints. To represent these degrees of freedom, we define an equal number of *control parameters* that control the configuration and motion of the end-effector. The set of valid control parameters consists of x, y, z, ψ, ϕ and θ, where x, y, z define translations parallel to the respective axis and ψ, ϕ, θ define rotations about the X-, Y- and Z-axis, respectively.

As an example, consider the three-link planar arm shown in Fig. 4, with three rotational joints in the XZ-plane. The transform ${}^{B}_{T}T$, describing the configuration of the end-effector relative to the base, is dependent on the actual construct that links the end-effector to the base. Although it is possible to represent exactly the end-effector's configuration in terms of the joint parameters ϕ_1, ϕ_2 and ϕ_3 and link lengths $x_1 = l_1$ and $x_2 = l_2$, it is worthwhile to explore alternative approximate representations, especially from considerations of user control (of the simulation) and computational efficiency.

A simple examination reveals that the workspace is bound by an outer circle with radius $l_1 + l_2$, and that the end-effector can assume any orientation in the XZ-plane (Fig. 5(a)). A very crude approximation (but one which may prove useful) of these characteristics can be achieved using a single rotational parameter ϕ and two independent translational parameters x and z. Given the values x_{min}, x_{max}, z_{min} and z_{max},

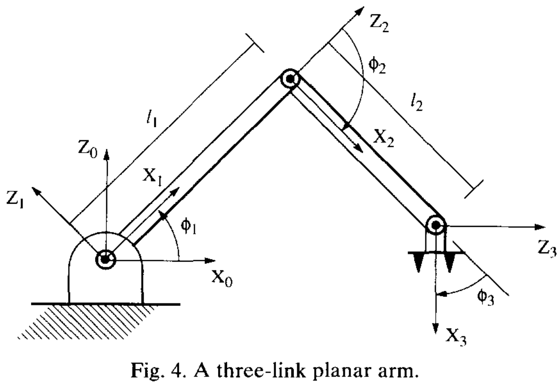

Fig. 4. A three-link planar arm.

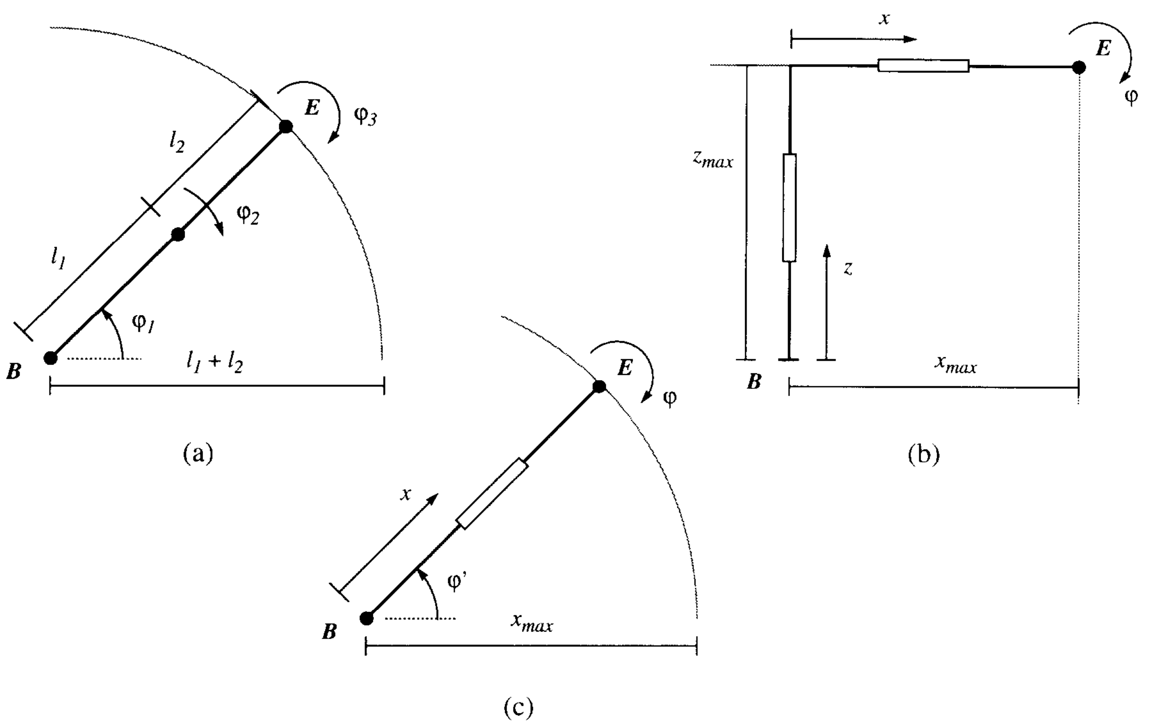

Fig. 5. Control parameters and workspace for different approximations of the three-link planar arm.

specifying the value ranges for the control parameters x and z, the resulting workspace is bound by a rectangle instead of a circle (Fig. 5(b)). Depending on the value ranges specified, the circle with radius $l_1 + l_2$ may inscribe, intersect or circumscribe the rectangle. This approximation can be refined by introducing an additional rotational parameter instead of one of the translational parameters. This results in a specification of the end-effector's position using polar coordinates x and ϕ' (Fig. 5(c)). A further refinement would yield the exact representation.

2.3. Motion rules

Often, the control parameters reflect the joint parameters that specify the configuration of the joints, and thus of consecutive links. However, when describing the configuration and/or motion of a mobile robot's base with respect to the outside world, the control parameters take a slightly different physical meaning. In most cases, the motional capabilities are defined locally, dependent on the current configuration of the body. In slight contrast, the configuration itself is defined globally, relative to a fixed reference frame. Therefore, even though infinitesimal small motions can be readily described using one representation, the global result is not restricted to that same representation, nor is the configuration of the body. The obvious example of such a discrepancy between configuration control and motion control is a "car". We refer to Barraquand and Latombe [5] for an "exact" solution, i.e., motion-planning algorithm, for the "car problem". For the sake of simplicity we assume that the "car", or mobile agent, is capable of stationary rotational motion.

Thus, the configuration may become practically independent of the motion mechanism, while, at the same time, the motion control parameters may only approximate the mechanism's complexity. To represent these differences in configuration and motion control we introduce a set of *motion rules* that are used to update the values of the control parameters, especially in the case when the parameters represent only the configuration and not the motional capabilities of the robot.

The objective for specifying motion rules is not only to solve this discrepancy between configuration and motion control, but also as a tool to embody the discretization of time. The simulation of a construction process relies on an *activity network* of construction activities and an accompanying time-history state representation of the building under construction [6]. The simulation of the activities and of the evolving geometry require a time step to be specified. This time step defines the discretization of the simulation process and of the robot's motion. The resulting motion steps also depend on the robot's joint velocities. In reality, these velocities may vary in time in order to achieve a certain smoothness in the motion (see [4] for a definition of *smooth*). Since, in simulation, motion is not represented as a continuous but as a discrete function, we may assume the velocity to be constant, but possibly dependent on the robot and/or joint type. We further disregard the velocity as a parameter in the process and use instead a set of motion step values, one for each control parameter, that are denoted x_s, y_s, z_s, θ_s, ϕ_s, and ψ_s. These values are specified except for their sign (positive or negative). Within a set of motion rules each of these values may take either sign, but this sign has to be the same for all rules in the set.

We define a *transform function*, with respect to a set of arguments, to be any series of terms composed with addition and/or multiplication, where each term may be either an argument, a *sin* or *cos* of an argument or a negation of any of the above. Then, a motion rule takes on the following form: $p \leftarrow f(\{a_1, a_2, \ldots, a_n\})$. Here p denotes a control parameter and f denotes a transform function; $\{a_1, a_2, \ldots, a_n\}$ denotes a set of arguments to f, chosen from the set of control parameters describing the body's configuration and the sets of step and constant values. Such a motion rule generally defines the value of p at time $t + \Delta t$, where Δt denotes the time step, in terms of the control parameter's values at time t and the step and constant values. In particular, these motion rules can be used in three different ways, depending on the arguments specified to the transform function. Firstly, a motion rule may specify a constant value to a parameter. In this case, the transform function takes a single argument, which should be the constant value corresponding to the control parameter, e.g., $x \leftarrow x_c$. Even though, strictly speaking, these are not "motion" rules, such degenerate rules are useful in order to achieve a uniform representation. The constant values, one for each control parameter, are denoted x_c, y_c, z_c, θ_c, ϕ_c, and ψ_c. They may also serve as initial values for the configuration.

Secondly, the motion rule may specify the motion step for the control parameter. Here, the transform function takes two arguments, which are the control parameter itself and the corresponding step value, e.g., $x \leftarrow x_s$. Thirdly, in the general case, a motion rule may specify the parameter's new value in terms of other control parameters, e.g., $x \leftarrow x + r_s \cos\theta$.

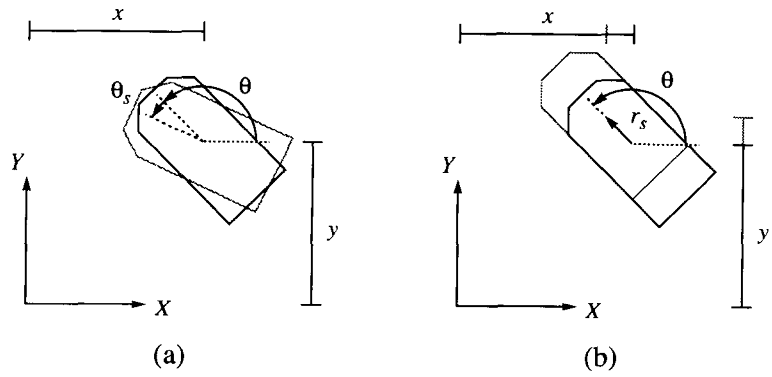

Fig. 6. A mobile agent with (a) stationary rotational motion and (b) translational motion along its axis.

As an example, consider a mobile agent with stationary rotational motion and translational motion along its axis (Fig. 6). The agent has control parameters x, y, z, and θ. However, its motion is specified in polar coordinates. Thus, the resulting motion rules for a horizontal work surface are

$$\{\theta \leftarrow \theta + \theta_s, x \leftarrow x + x_s \cos\theta - y_s \sin\theta,$$
$$y \leftarrow y + x_s \sin\theta + y_s \cos\theta, z \leftarrow z_c\}.$$

3. Motion planning

In path planning we distinguish between global and local methods. Global techniques are generally based on *configuration space* (*C-space*) (due to Lozano-Pérez and Wesley [7]) and have been demonstrated successfully. However, finding collision-free paths in three dimensions and for objects with rotational degrees of freedom increases the dimensionality of the configuration space significantly, and increases the computation time exponentially [5]. Further, if the environment changes dynamically, as is the case in building construction, this requires a dynamically varying C-space, which is computationally inefficient. This will detract from some of the advantages offered by the global techniques. Local strategies may function both in task space and in configuration space. They are generally based on artificial potential functions [8], but exhibit a limitation that is due to the appearance of local minima.

In order to deal with a dynamically changing environment it is often useful to follow a local approach where obstacles are avoided as they are encountered [9]. In this approach a path is built up from single motion steps in a bottom-up fashion. We denote this approach *motion planning* to distinguish it from global path planning. Such an approach may be combined with an approximated global path finding algorithm into a hybrid method. Ilari and Reyna [10] and Ilari and Torras [11] adopt a hybrid method with a global path planning phase and a solution path search phase, the latter based on heuristics.

Our objective is to discuss the rudiments of a general three dimensional motion planning algorithm that is based on the robot representation described above. This motion planning may be either goal-driven or path-driven. The latter constitutes a hybrid method where the motion planning is influenced by a global path chosen a priori. Due to the dynamically changing environment this global path may not be a sufficient solution, but constitutes a guiding path for motion planning. In the discussion that follows the term goal may be interpreted either as the final goal or as any point on the global path that is temporarily designated the goal configuration.

3.1. Motion planning algorithm

We define motion planning as the production of a plan that depends upon (i) the distance from the current position to the goal position, and (ii) the required orientation at the goal relative to the current orientation. The plan is constrained by the obstacles surrounding the current position, and any specified motion heuristics (see [10], [11]). We base our motion planning algorithm on the robot representation specified above, and in particular on the motion steps allowed and on the workspaces of both the base and the end-effector of the robot.

Lozano-Pérez and Wesley [7] also describe a simple collision avoidance algorithm for path planning based on the "generate and test" paradigm. Adapted to motion planning the algorithm becomes:

(1) compute the volume swept out by the moving object for the proposed motion step,
(2) determine the intersection of the swept volume and the surrounding obstacles,
(3) if an intersection exists, propose a new motion step.

To determine whether two objects intersect, Brooks [12] introduces the idea of generalized cones, and Bonner and Kelley [13] introduce Successive Spherical Approximations as a complete model for three dimensional objects, in order to approximate free space at successive levels of detail. Krishnamurti ([14], [15]) develops an algorithm for shape intersection based on a representation of three dimensional polygons or polyhedra in terms of maximal spatial elements. Since the number of possible motion steps is limited, it is straightforward to determine the swept volume of the object and of the robot for each motion step.

3.2. Motion language

It remains to be specified which motion steps should be proposed and in what order. This is equivalent to asking which motion step is most appropriate at each moment or for each configuration, given the constraints that adhere to the current situation. Such a decision is highly dependent on the particular robot and on its expected behavior. The motion language serves as the means to specify this behavior and, in particular, the order in which the allowed motion steps should be proposed.

The key to a behavioral description of a robot is to specify exactly what behavior is expected of the robot for each situation that may be encountered. This requires a situation-based decomposition of the behavioral description, the equivalent of which is a case-structure in a programming language; a selection is made depending on a particular value or condition. We have adopted a *which*-structure with *labels* that mark the different behavioral modules, where each label reflects the situation that is handled within that module. Examples of such situations are whether the robot is transporting material or not, whether safety considerations are at the brink of being violated, etc.

Each module specifies a series of actions, each of which is subject to verification of its validity in the light of the surrounding obstacles. We have refrained from using an explicit *if-then-else*-structure. Instead, commands (motion steps) are "*chain*ed" together; each element in the chain is weaved in with the previous and the next element. If the current command fails the next one is invoked, but whenever possible an attempt is made to try the previous command. That is, commands positionally later in the chain are invoked only when necessary. This results in a procedure as outlined below. The procedure ends when either the goal is reached or the end of the chain is encountered.

```
1 if the current command succeeds
2   then if the goal has been reached
3          then exit
4        if there is a previous command in the chain
5          then current ← previous
6   else if there is a next command in the chain
7          then current ← next
8          else exit
9 goto line 1
```

An extension to this concept is the *cycle*, or cyclic chain. Here, no end exists to the chain, only a starting point, but the chain is augmented with a conditional statement that specifies when to exit the cycle. This construct is particularly useful when the behavior should be one of following a particular (open or closed) boundary or path. The exiting condition then may reflect a spatial decision crossroads, such as the end of an open boundary (or path) or the point of bifurcation with respect to a specific goal (or subgoal). A final control element of the language is the specification of a set of commands of which only one is active at any time. Only on failure is such a command replaced by the next command in the set. This control structure is denoted *each*.

The elementary building blocks of the language are the motion commands *move_to* and *move_dir* together with a series of functions that may be used as command arguments. Assignments are always specified in conjunction with the *var* operator that associates a variable to its argument, which is typically a control parameter.

commands:

- **move_to** (*parameter*, *goal value*)
  A single step in the direction of the goal is performed for the specified parameter, if allowable. If the goal value equals the current value, no step is performed. Returns TRUE if the motion step succeeded, else returns FALSE.
- **move_dir** (*parameter*, *direction value*)
  A single step in the specified direction is performed for the specified parameter, if allowable. Returns TRUE if the motion step succeeded, else returns FALSE.

assignment:

- **var** $(x) = value$
  The specified value is stored in variable x.

functions:

- **goal** (x)
  Returns the goal value of parameter x.
- **r_goal** (x)
  Determines the goal value for parameter x relative to the current configuration.
- **dir** (x, v)
  Determines the direction specified by v with respect to the current value of parameter x.
- **axis** (x, v)
  Reduces the angle v to between $-90°$ and $90°$ with respect to the axis defined by the current value of x.
- **min** (x); **max** (x)
  Returns the minimum (maximum) value specified for parameter x.
- **min** (x, v); **max** (x, v)
  Returns the minimum (maximum) of the current value of x and the value v.
- **var** (x)
  Returns the value of the variable associated with parameter x.

4. Examples

We consider three examples: an exterior wall finishing robot, an automated crane hoist and an automated mobile agent. The first one constitutes a simple example illustrating the issues involved when describing a robot's behavior. The second example demonstrates the usage of the behavioral language in describing the behavior of a robot agent. The final example explores the description of the more complicated behavior of a mobile agent.

4.1. Exterior wall finishing robot

Warszawski [16] distinguishes this type of robot for usage in finishing activities on building exteriors such as painting, plastering and finish inspection. A possible robot configuration is illustrated in Fig. 7. It consists of a vertical carriage suspended from a base vehicle located on the roof. The base allows for horizontal translational motion along the building edge. The carriage has the ability to move vertically along the building exterior surface. The robot's end-effector also has limited horizontal displacement capability that allows it to cover a vertical strip of the surface without moving the base.

Firstly, consider the base as an automated agent. Its motional capability is sufficiently described by a single control parameter x; the corresponding motion rule is $x \leftarrow x + x_s$. The behavior of the base as an automated agent consists of translational motion in order to reach the x-coordinate of the overall goal position.

$chain\{move\_to(x, goal(x))\};$

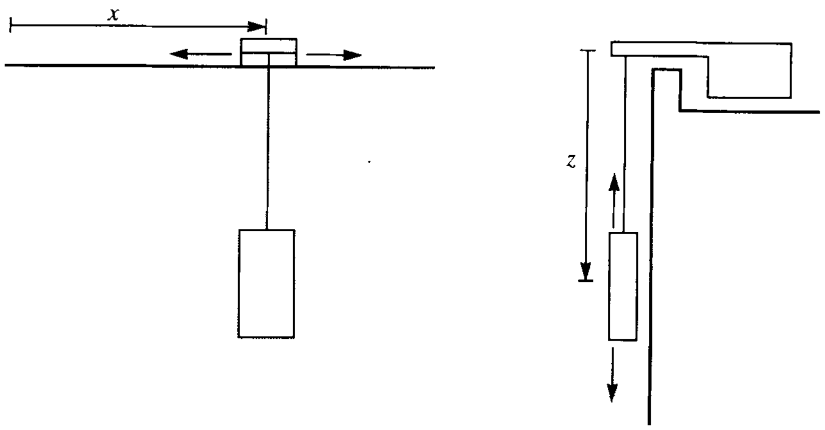

Fig. 7. Front and side view of a possible configuration of an exterior wall finishing robot.

Secondly, consider the carriage as an automated agent independent of the base of the robot, initially, assuming a fixed end-effector. In this case, the motional capability is described using the control parameter z, representing the allowable vertical displacement which is defined relative to the base atop the roof. Thus, the behavior of the carriage (with fixed end-effector) consists only of vertical motion in order to reach the z-coordinate of the overall goal position.

```
chain{move_to(z,goal(z))};
```

If we now consider the end-effector as capable of horizontal motion, its displacement can be defined relative to the carriage. The behavior of the robot's end-effector (taken as an automated agent independent of the base vehicle) is identical to the behavior of the base agent. Upon considering their dependency, a choice as to whether the base or end-effector should be moved can be made depending on the goal location and the reach of the end-effector, where the latter is specified relative to the carriage location by values x_{min} and x_{max}.

4.2. Automated crane hoist

We consider the motion of an automated crane hoist with fixed base and full-site access (see Fig. 8). In general, a crane hoist's end-effector has only three degrees of freedom. These are represented exactly by the control parameters θ, x and z, where θ represents the orientation of the crane hoist, and x and z represent the respective horizontal and vertical displacement of the end-effector with respect to the base. In the case of a crane hoist with a limited work envelope, the values θ_{min}, θ_{max}, x_{min}, x_{max}, z_{min} and z_{max} (relative to the base position) are specified to define the workspace of the hoist. The corresponding motion rules are $\{\theta \leftarrow \theta + \theta_s, x \leftarrow x + x_s, z \leftarrow z + z_s\}$. When applying these rules, the values of θ_s, x_s and z_s may be either positive or negative.

Given any obstacle, the expected behavior of the crane should be to raise the load above the height of the obstacle, if possible, and complete its trajectory subsequently, only lowering the load when the goal position has been reached. A corresponding motion plan for an automated crane hoist is shown below.

```
set the goal height
var(height) = 30;
move clear of the goal height
chain {
  move_to (z, max(z, var(height) + goal(z)))
};
move closer to the base
chain {move_to (x, min(x, goal(x)))};
alter the orientation towards the goal
chain {
  move_to (θ, goal(θ)),
  while staying clear of any obstacles
  move_dir (z, dir(z, max(z)))
};
alter the displacement towards the goal
chain {
  move_to (x, goal(x)),
  while staying clear of any obstacles
  move_dir (z, dir(z, max(z)))
};
which {
  transport:
    when transporting an object
    stop clear of the goal
    chain {move_to(z, var(height) + goal(z))};
  default:
    else
    lower the end-effector to the goal height
    chain {move_to (z, goal(z))};
};
```

In many cases it may be either impossible to raise the load above an obstacle, or it may be unacceptable for safety reasons when a human labor crew is working underneath the planned trajectory. Then, lateral displacement as a result of rotational motion should be considered as a way to circumvent such "obstacles". The following motion sub-plan proposes lateral motion:

```
try a combination
chain {
  of translational motion
  var(x) = dir(x, goal(x)),
  and rotational motion
```

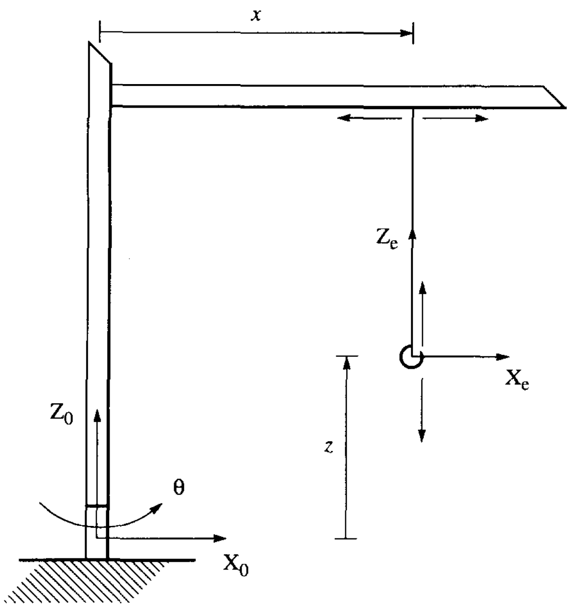

Fig. 8. A crane hoist.

```
    move_to ($\theta$, goal($\theta$)),
    the former in either direction but not intermixed
    each {move_dir ($x$, var($x$)),
          move_dir ($x$, $-$var($x$))}
};
```

A choice in direction is made depending on the initial direction of the goal with respect to the end-effector, and any motion is continued in the same initial direction, unless prohibited by the obstacle.

4.3. Automated mobile agent

Consider the mobile agent illustrated in Fig. 6. It has the ability to move horizontally through stationary rotational motion and through translational displacements along its axis. the corresponding control parameters are x, y, z and θ and the motion rules are $\{\theta \leftarrow \theta + \theta_s,\ x \leftarrow x + x_s \cos\theta - y_s \sin\theta,\ y \leftarrow y + x_s \sin\theta + y_s \cos\theta,\ z \leftarrow z_c\}$.

We assume different behavior depending on whether it is transporting an object or not. In the former case, translational forward motion is preferable over backward motion. In the latter case, no distinction is made between forward and backward. That is, when considering a rotation of the axis of the mobile agent towards the goal, the axis is treated as undirected. Overall, the agent is oriented towards the goal, if possible, in order to shorten the necessary trajectory. If an obstacle is found on its path toward the goal, the agent follows the obstacle's boundary until it can resume a straight path towards the goal.

```
var($\theta$) = dir($\theta$, r_goal($\theta$));
which {
  transport:
    when transporting an object
    try to orientate itself relative to the goal
    chain {move_to ($\theta$, r_goal($\theta$))};
    var($x$) = dir($x$, r_goal($x$));
    keep an orientation towards the goal
    while the goal has not been reached
    chain {move_to ($\theta$, r_goal($\theta$)),
           move_to ($x$, r_goal($x$)),
           move_dir ($\theta$, dir($\theta$, r_goal($\theta$))),
      follow the boundary of the obstacle by
      using a combination of the following
      moves:
        move forward
        move_dir ($x$, var($x$)),
        rotate one way
        move_dir ($\theta$, var($\theta$)),
        move backward
        move_dir ($x$, $-$var($x$)),
        or rotate the other way
        move_dir ($\theta$, $-$var($\theta$))};
default:
  else
  follow the same behavior but assume the axis
  is not oriented
  chain {move_to ($\theta$, axis($\theta$, r_goal($\theta$)))};
  var($x$) = dir($x$, r_goal($x$));
  chain {move_to ($\theta$, axis ($\theta$, r_goal($\theta$))),
        move_to ($x$, r_goal($x$)),
        move_dir ($\theta$, dir($\theta$, r_goal($\theta$))),
        move_dir ($x$, var($x$)),
        move_dir ($\theta$, var($\theta$)),
        move_dir ($x$, $-$var($x$)),
        move_dir ($\theta$, $-$var($\theta$))};
};
finally, adopt the correct orientation
chain {move_to ($\theta$, goal($\theta$))};
```

5. The RUBICON simulator

The findings of this research have been implemented in a simulation program, RUBICON. The program can be used to study different task plans, report on different robot types, study alternate robot–human crew mix examples, and produce measurements in terms of time or cost. Working examples include an automated crane hoist and an automated tow-motor active in the construction of a typical precast concrete residential building. The results of the simulation will assist an engineer or planner to decide on the appropriate mix of robots and human labor crews in the planning stage of a building construction project.

5.1. RUBICON

The simulation takes a building construction task plan as input. This is a detailed plan describing the construction elements and the construction process as a task schedule. A task is to be performed either by a human crew or by a robot. The simulator then translates each task into a robot motion plan (a sequence of motion steps) using a rule-based description of the robot agents. The motion plans reflect the respective robot's motional capabilities and limits, and avoid any collision. The simulation also rejects impossible tasks and allows for a variety of different robot types. The result of the simulation consists of a graphical visualization of the motion plans, and of the building under construction.

The input to the RUBICON program consists of two kinds of files. One file contains the task plan [17] describing the components and the construction process as a sequence of tasks, each of which is performed by a human or robot crew. For each robot agent specified in the task plan, a motion file is read in that contains a description

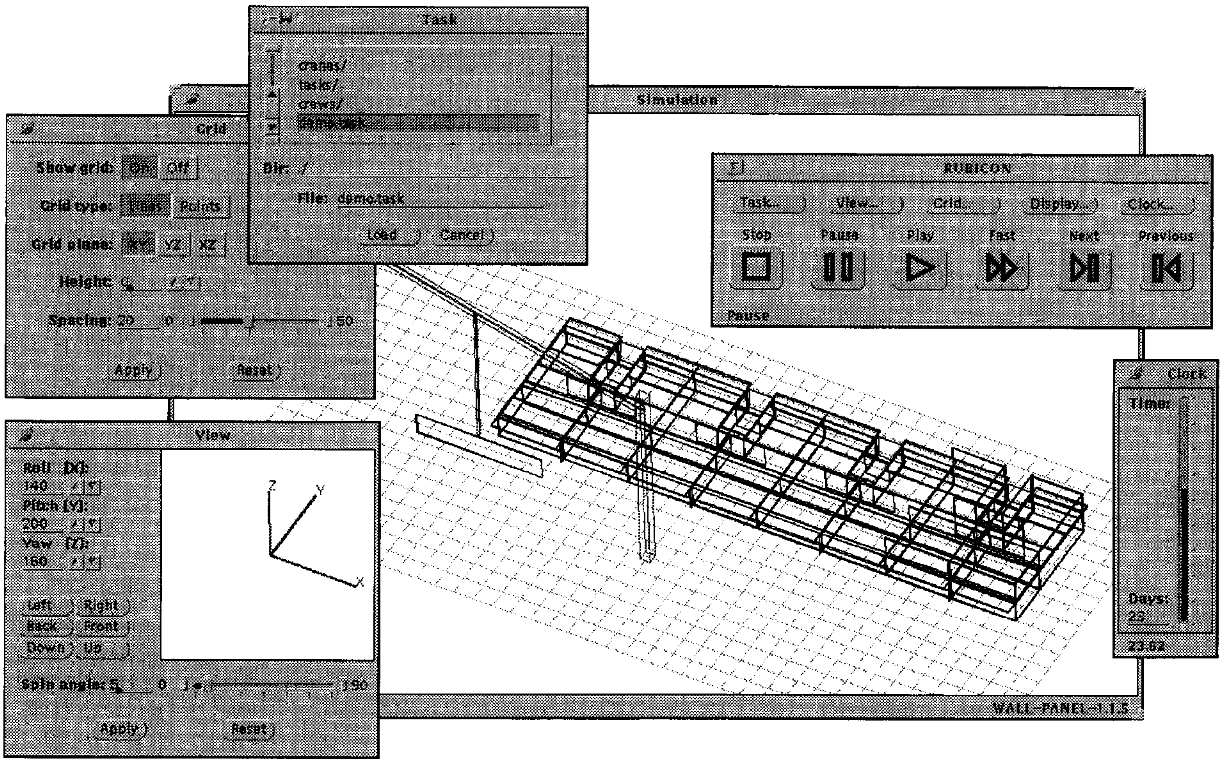

Fig. 9. Snapshot of the RUBICON graphical user interface.

of the motional capabilities of the robot and the motion rule set describing its intended behavior, expressed in the motion language specified above.

The output of the RUBICON program is a graphical simulation of the construction process as specified in the task plan, with a visualization of the motional actions of the robot agents and of the transportation of the construction compo-

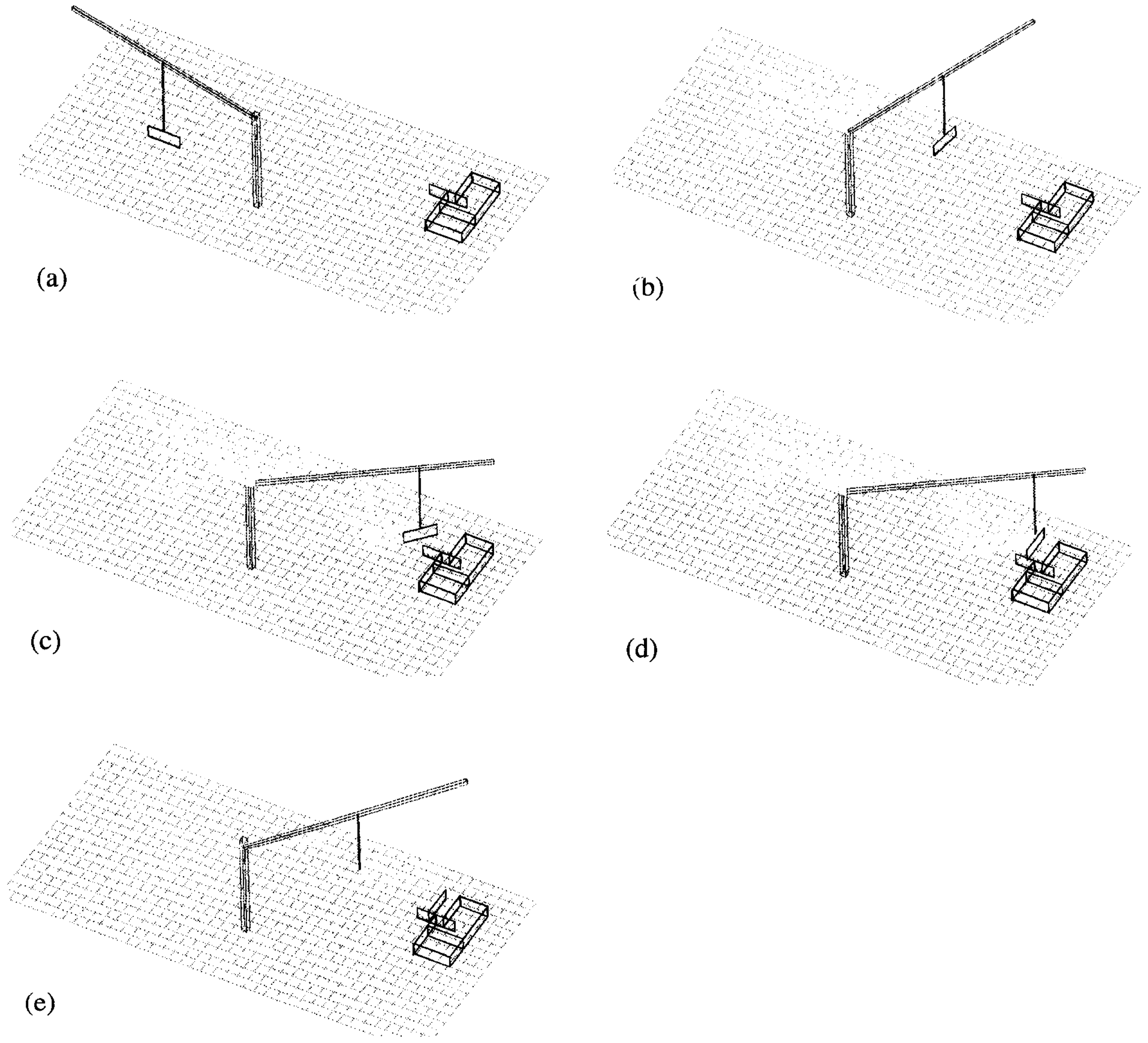

Fig. 10. Snapshots of the simulation: (a) Step 1: The panel is picked up from the truck site or delivery location, after a human crew has taken care of connecting the panel to the crane hoist's tool. (b) Step 2: The robot crane uses rotational motion (about its axis) to move the panel from the delivery location closer to its final position. Obstacles are avoided by raising the panel. (c) Step 3: The robot crane uses (radial) translational motion to move the panel above its final position. Obstacles are avoided by raising the panel. (d) Step 4: While still attached to the robot's tool, the panel is rotated to its final orientation by a human crew. (e) Step 5: Aided by the human crew, the panel is lowered to its placement location and disconnected from the robot's tool. The robot crane returns to the truck site for its next task.

nents by these agents, and with a specification of the process time. The output is controlled through an audio/video-like control panel with buttons for *play*, *fast* play, *next*, *previous*, *pause* and *stop*. *Fast* play is achieved by unmaterializing the robot agent; *next* and *previous* instantly jump to the next, respectively previous, construction component.

Other interface panels allow the user to choose and load a new task plan and corresponding motion files, alter the 3D-viewing parameters, alter the displayed grid, and view the current process time (see Fig. 9).

5.2. Demonstration of the program

Figure 10 illustrates an exemplar simulation of a single task: the placement of a wall-panel in the second unit of the first floor of the building under construction.

5.3. Simulation results

The simulation can be used to make observations on the feasibility of each task and to maintain a running measure of construction time and cost. It can also be used to study the productivity of alternate construction plans and alternate resource mixes or robot types. Examples of simulation results are demonstrated in [17]. RUBICON may serve as a tool for construction company engineers to perform studies on real project task plans; engineers/developers may use RUBICON to develop better task planners.

6. Conclusion

In this paper we have outlined the uniform representation for a variety of robot agents for use in a potentially real-world application, namely the use of automated agents in cooperation with humans to assembly building. We have introduced motion rules as a way of capturing the diversity between the different robot types in a uniform manner, and we have shown how these motion rules can be incorporated into a general three dimensional motion planning algorithm based on the "generate and test" paradigm.

We have presented some basic structural, operational and functional elements of a motion language that is used to describe the behavior of a robot in a multitude of different situations. The resulting motion plan specifies a time-dependent trajectory as a series of motion steps that is particular to the situations encountered along the trajectory.

The simulation program, RUBICON, is demonstrated for a residential building example constructed with precast concrete panels.

Three simple robot examples are described in the paper, two of which were used in our simulation studies: a crane for moving precast concrete panels and a tow-motor to position palletized material.

Finally, it is interesting to draw a comparison between the approach presented in this paper to simulate building construction and generative approaches to building design. Both are goal driven. Both are rule based. Both seek to arrive at sequences of "admissible" steps. The essential difference is in the vocabulary which, in our case, includes the sets of building blocks and the sets of robot agents.

Acknowledgements

The work reported in this paper was funded in part by the Japan Research Institute, Tokyo.

References

[1] L. Petropoulakis and C. Malcolm, Programming autonomous assembly agents: functionality and robustness, DAI Res. Paper 468, Dept. of Artificial Intelligence, University of Edinburgh, 1990.
[2] C. Malcolm, Planning and performing the robotic assembly of some cube constructions, M.Sc. Dissertation, University of Edinburgh, 1987.
[3] C. Malcolm, T. Smithers and J. Hallam, An emerging paradigm in robot architecture, *Proc. 2nd Conf. on Intelligent Autonomous Systems*, 1989.
[4] J.J. Craig, *Introduction to Robotics: Mechanics and Control* (2nd ed.), Addison-Wesley, Reading, Mass. (1989).

[5] J. Barraquand and J.-C. Latombe, Robot motion planning: a distributed representation approach, STAN-CS-89-1257, Dept. of Comp. Science, Stanford University, 1989.

[6] C. Hendrickson and T. Au, *Project Management for Construction: Fundamental Concepts for Owners, Engineers, Architects and Builders*, Prentice-Hall, Englewood Cliffs, N.J. (1989).

[7] T. Lozano-Pérez and M.A. Wesley, An algorithm for planning collision-free paths among polyhedral obstacles, *Comm. ACM*, *22* (1979) 560–570.

[8] O. Khatib, Real-time obstacle avoidance for manipulations and mobile robots, *Int. J. Robot. Res.*, *5* (1986) 90–98.

[9] V.J. Lumelsky, Continuous motion planning in unknown environment for a 3D cartesian robot arm, *Proc. IEEE Int. Conf. on Robotics and Automation*, 1986, pp. 1050–1055.

[10] J. Ilari and J.Ll. Reyna, Some experimental results using heuristics for solving the find-path problem in C-space, *Theory of Robots. Selected Papers from the IFAC / IFIP / IMACS Symp.*, 1986, pp. 337–342.

[11] J. Ilari and C. Torras, The classical 2D find-path problem: improving search efficiency by using orientation heuristics, *Mobile Robots II. Proc. SPIE* 852, 1987, pp. 248–255.

[12] R.A. Brooks, Solving the find-path problem by good representation of free-space. *IEEE Trans. Systems, Man and Cybernet.*, *13* (1983) 190–197.

[13] S. Bonner and R.B. Kelley, Planning 3-D collision-free paths, *IEEE Int. Symp. on Intelligent Control*, 1989, pp. 550–555.

[14] R. Krishnamurti, The maximal representation of a shape, *Environ. Planning B: Planning and Design*, *19* (1992) 267–288.

[15] R. Krishnamurti, The arithmetic of maximal planes, *Environ. Planning B: Planning and Design*, *19* (1992) 431–464.

[16] A. Warszawski, Robots in the construction industry, *Robotica*, *4* (1986) 181–188.

[17] R. Stouffs, R. Krishnamurti, S.R. Lee and I.J. Oppenheim, Construction process simulation with rule-based robot path planning, *Automat. Construction, 3* (1994) 79–86.