A Control Architecture for Robotic Excavation in Construction

Q. P. Ha*

ARC Centre of Excellence in Autonomous Systems (CAS), Faculty of Engineering, The University of Technology, Sydney, PO Box 123, Broadway 2007 NSW, Australia

&

D. C. Rye

ARC Centre of Excellence in Autonomous Systems (CAS), Australian Centre for Field Robotics, The University of Sydney, Rose Street Building J04, 2006 NSW, Australia

Abstract: This article presents a hybrid control architecture developed for robotic excavation. The lower-level controllers are designed using a combination of sliding mode control and fuzzy logic control. Control strategies at the higher level involve task decomposition in association with statecharts, and task execution and verification. Typical machine tasks are decomposed into subtasks and/or states. Graphical notation is introduced to facilitate software implementation of the control flow within statecharts. For each subtask, a statechart via its states at different priority levels selects to activate a number of corresponding lower-level controllers as a result from resolution of the machine resources and the control and data flows. The control architecture is designed with a view to managing hierarchical complexity, and facilitating the application of formal verification methods, software reuse, and lower-level control results. Field test results through the task of digging a trench are provided to demonstrate the feasibility of robotic excavation in moving toward construction automation.

1 INTRODUCTION

The past two decades have witnessed some successful applications of advanced robotics and automation technologies in construction. Intense competition, shortages of skilled labor, and technological advances are forcing rapid change in the construction industry, motivating increased attention on construction automation. Construction typically involves a number of earthmoving machines including excavators. Common excavation operations in construction include general earthmoving, digging, and sheet piling for removing large amounts of material, alongside the precisely controlled actions required for trenching and footing formation.

Despite acknowledgment of the rapid progress of robotics and automation applications in construction (Gamboa and Balaguer, 2002), there have been few actual implementations of robotic excavators. Much of the work on terrestrial excavation has focused instead on teleoperation. Although a number of valuable theoretical and experimental contributions have been made by academics in the field of robotic excavation (see, for example, Singh, 1997; Bradley and Seward, 1998; Ha et al., 1999), autonomous operation of a full-scale excavator has not been commercially demonstrated to date.
The application of robotic technology and computer control is an important key to construction industry automation. The usual task of a backhoe excavator is to loosen and remove material from its original location, and to transfer it to another location by lowering the bucket, digging by dragging the bucket through the soil, then lifting, slewing, and dumping the bucket load. An autonomous excavator can be regarded as a robot having its arm not fixed relative to the work piece. It plastically deforms the work piece (soil) by applying large shear forces, and at the same time is caused to move relative to the soil by the same large forces. Strategic dig planning and bucket trajectory planning remain important issues in such a dynamic environment as a construction site. While performing a robotic task, the excavator must change in an appropriate way the profile of the soil being worked. Moreover, the high variability in soil conditions encountered in construction makes robotic excavation very much more difficult than conventional robotic tasks. Excavation automation is thus a multidisciplinary task encompassing a broad area of research and development: planning, monitoring, environment sensing and modeling, navigation, and machine modeling and control. In addition to the axis control problem at the lowest level (Ha et al., 1999), task planning and execution at higher levels are also important considerations in autonomous excavation. Traditionally, a planner is used to determine a sequence of primitive actions that, when executed, will transform the world from an initial state to a goal state. Planning in general is thought to be hierarchical. Planning, according to Arkin (1998), should also be deliberative as it requires relatively complete knowledge of the world and uses this knowledge to predict the outcome of its operations.

This article presents a control architecture for a mini-excavator that is retrofitted as an 8-DOF robotic excavator, as described in Le et al. (1998). At the lower level, fuzzy logic control is used to encapsulate expert experience for capturing soil in difficult digging conditions, in conjunction with sliding mode control to improve robustness against control uncertainties. Superficially, the key control objective is to fill the bucket in the shortest possible time. The higher-level control involves behavioral/hierarchical control schemes, which are based on the decomposition of tasks of the robotic machine into subtasks or states. This approach provides the flexibility to execute complex task spaces using Unified Modeling Language (UML) capsules and statecharts.

The remainder of the article is organized as follows. Section 2 introduces the experimental robotic excavator and its equipment, whereas Section 3 describes the proposed architecture framework. A rationale for the decomposition of excavation tasks is presented in Section 4. The structure of higher- and lower-level control is addressed in Section 5 and field-test results are presented in Section 6. Finally, a conclusion is given in Section 7.

2 EXPERIMENTAL ROBOTIC EXCAVATOR

A 1.5-ton hydraulically actuated mini excavator is used as the basis for our experimental work. This machine has been extensively modified to produce a robot having eight hydraulic actuators: right and left travel motors, a cab slew motor, and two-way hydraulic cylinders on the boom swing, boom, dipper arm, bucket, and back-fill blade axes. Electrohydraulic servovalves are installed, replacing the original manually actuated direction control valves. Ancillary equipment added to support the servovalves include an accumulator with unloading valve, solenoid check valves, and an oil-to-air radiator.

The experimental excavator is extensively instrumented with a number of machine-state and environment sensors. The robotic arm joint angles are measured using absolute encoders. The hydraulic system is instrumented with transducers that measure the actuator pressures and the valve spool positions. Strain-gauge force sensors enable direct force measurement during digging. A commercial time-of-flight laser measurement system is used to scan the terrain on one side of the bucket, providing a surface profile with 10 mm resolution and a statistical error of ±15 mm within a sensing range of 1–8 m. Control is achieved through digital controllers in conjunction with an industrial PC. Figure 1 shows the sensing and control hardware; loop closure is achieved within the industrial PC, with the digital controllers providing interface and safety monitoring functionality. The system is fully self-contained, with power derived from the excavator’s electrical system.

In addition to data obtained by machine sensors, estimation of some inaccessible states is required for control and monitoring purposes. In robotic excavation, it is also desirable to estimate the uncertain forces arising from interactions between the bucket and the soil, and other external disturbances such as friction and load inertia changes, so as to reject their influence and achieve robust performance. For this, various estimation techniques can be used. In Ha et al. (2000), a variable structure systems approach to friction estimation and compensation has been applied to electrohydraulic servo systems for external disturbance rejection in force-and-position control of the robotic excavator.

3 ARCHITECTURE FRAMEWORK

Our proposed architecture will be embodied as capsule and statechart objects within the UML framework for
real time object-oriented modeling (ROOM) detailed in Selic et al. (1994). This framework provides a suitable abstraction level and well-defined task interfaces. Some necessary terminology is introduced below:

**Capsules** are complex, potentially concurrent, and possibly distributed active architectural components that interact with their surroundings through one or more signal-based boundary objects called ports. A port is a part of the capsule implementation that interfaces with other capsules. Ports realize protocols, which define the valid flow of information (signals) between connected ports of capsules. Requiring capsules to communicate only through ports allows for fully encapsulating their internal implementations from the environment. Ports can be conjugated to show the incoming and outgoing directions. **Connectors** capture the key communication relationships between capsules. **Statecharts** are used to implement the functionality of simple capsules. More complex capsules combine the statecharts with an internal network of collaborating sub-capsules or sub-tasks joined by connectors. The state-machine, the sub-capsules, and their connections network represent parts of the implementation of a capsule. Figure 2 shows the proposed machine task architecture with the following capsule types:

**GUI**: Graphical User Interface that allows for interactions with the user.

**Expert**: This capsule will provide all information regarding decision-making, planning, and execution of tasks according to the goal defined by the user. Additionally, this capsule will involve activation of axis controllers that are required for execution of a task. It will also make recommendations to the conflict resolution capsule on how a particular conflict may be resolved.

**Conflict resolution**: Concurrent execution is supported with arbitration provided by the conflict resolution capsule. Different types of conflict resolution approaches (command fusion and command arbitration algorithms)
Machine Conflict Resolution Sensor Processing Safety Management Expert GUI

Fig. 2. Machine task architecture. (Capsules are represented by the bold rectangles, with ports shown as the light squares at the capsule boundaries.)

can be found in Pirjanian (1999) and the references therein.

Sensor processing: This capsule involves sensor signal conditioning, filtering, and any processing required.

Safety: The safety capsule monitors a health message (ON and OK) from all activated object tasks. If any active task fails to transmit this message the safety coordinator cannot send its health message to the safety capsule. The safety capsule will set safe values to the output vector.

The machine system developed can include later communication modules to send its information in a network to communicate with other machines. The basic task structure can be seen as a task with one upper-level interface and zero or lower-level task interfaces, as shown in Figure 3. The structure comprises two capsule types:

Task supervisor: This capsule will provide a mapping of the task control policies from the subtasks, based on the control input received through the upper level task interface. The capsule will resolve these policies into commands for the subtasks that are under its control. It will also react to internal events that are sent from the subtasks. If its policies cannot be resolved by the subtask events, a request will be returned through the upper level task interface for proper handling. In some cases the task supervisor may require the help of another task of the same level or may need to send a safety issue to the safety module, using a fail-safe mechanism. All sensor information is received from the sensor interface. Information acquired can be used by the state-machine for trigger transitions, situation assessment, and controller activation.

Subtasks: These capsules will provide the subtask algorithms through one or more subtask interfaces. They will also help to resolve any control situation that does not require intervention by the task supervisor. For complex systems, these subtasks can be further decomposed into a subtask supervisor and a set of simpler subtasks.

This basic structure can functionally and recursively be decomposed to whatever depth is necessary, allowing for modeling of arbitrarily complex structures. A complex subtask is decomposed into simpler subtasks. When this decomposition is completed, a subtask can be expressed in terms of sequential and/or concurrent execution of atomic states. Note that there are two types of atomic states: one that interacts with the machine controllers and the other that cannot be made simpler by further decomposition.

Furthermore, our basic task structure can be used to provide levels of authority as presented in Figure 4. The highest level can be seen as the machine manager that has different subtask managers. The lowest level implements the machine controllers. The lower-level task interfaces are connected to a conflict resolution module and interact with the hardware actuators. The structure shown in Figure 4 is a static representation of executable tasks. A dynamic representation during run time will resemble a tree, to allow for creating, adding, deleting, and executing particular tasks.

Task dataflow is only permitted from level \(i\) to level \((i-1)\), within a same level, or between the task supervisor and subtasks by using message passing. The only exception is the safety messages that go directly to the safety capsule from any level.

This conceptual framework can be viewed as a three-dimensional task-composition space where \(X\)-axis represents task taxonomy (subtasks shown in Figure 3), \(Y\)-axis represents flow of control or subfunctions (recursion and statecharts), and \(Z\)-axis represents levels of authority (Figure 4). All tasks can be represented as nodes within this task-composition space.

Inheritance can be used to abstract some of the common classes that will be used to implement robotic...
tasks. For instance an abstract statechart may provide similar methods required for implementing some robotic functionality such as status and basic exception handling capabilities.

4 EXCAVATION TASK DECOMPOSITION

4.1 Rationale

In general, a task algorithm describes how the machine should attempt to reach a specified goal. In task planning, the task algorithm is decomposed into a number of (perhaps hierarchical) subtasks with a predefined order of execution (relative priority) by combining states or atomic states that do not need to be further decomposed. The basic philosophy behind the proposed control architecture is summarized as follows. The decomposition should be formulated such that it can cover all specifications of the task. It follows that a task can also be composed from subtasks and sub-states to form a super state (a state of nested states). The proposed architecture can be considered as a hybrid one combining top-down and bottom-up approaches. It allows for component reuse, and for building complex states if required.

For a descriptive notation, a task of a robotic system can be represented as a set of subtasks, with predefined entry conditions and exit conditions, and a priority or order of execution determined by a state-machine:

\[
\text{Task} = \text{tuple}\{\text{subtask}_1, \text{subtask}_2, \ldots, \text{subtask}_n\},
\]

entry-conditions, exit-conditions,

priority state-machine_{\text{main}}

Subtasks can be executed sequentially, in parallel, or both, depending on the information supplied to the state-machine and the priority. Concurrent state-machines can be seen as threads, so that relative thread priority will define the order of execution. The following basic operations applicable to tasks are defined:

Task union, \( T = T_1 \cup T_2 \), returns a task that is the union of tasks at a particular level. Entry-conditions, exit-conditions, and a state-machine for the new combination need to be supplied. Task unions help to build more complex tasks.

Task intersection, \( T = T_1 \cap T_2 \), returns a task that is the intersection of tasks at a particular level. Task intersections help to identify common subtasks in the static representation that can be instantiated on the dynamic representation (run-time objects).

Conditional task implies a condition in the form of a Boolean expression \( T = T_i \text{ if } x \), where the task \( T_i \) will be active only if the Boolean expression \( x \) holds.

In the same manner, some properties are defined:

Subset: Let \( ST_i \) and \( T_i \) be two tasks. Then \( ST_i \) is a subset of \( T_i \) if every element of \( ST_i \) is also an element of \( T_i \).
Equality: Let \( T_1 \) and \( T_2 \) be two tasks. Then \( T_1 \) is equal to \( T_2 \) if they have the same elements.

Cardinality (or size): It determines the number of different elements in a set. It can be used to count the number of subtasks within a defined task. Furthermore, task complexity can be measured as the number of states that are involved in a task.

Task scope resolution: It returns the absolute path to find the local state or subtask with respect to the expert main state-machine.

State-Machine ::= tuple{variables, status, current state, priority, entry-conditions, exit-conditions, substates, transitions,}

where,

\[
\text{Status} ::= [\text{Off, Idle, Run, Undefined, TimeOut, Exception}]\]

variables used in the state-machine can be seen as programming language variables, and “current state” is a special variable that keeps track of the current state of the state-machine. Priority indicates the execution order used when scheduling the subtask relative to other subtasks.

Transition is a special function that monitors the occurrence of an event. Functions can be defined as entry-conditions, code, or exit-conditions. A function contains the activated controller’s code and any algorithm required to execute the task at this abstraction level. The index “\( i \)” takes values from 1 to \( n \).

### 4.2 Task decomposition for robotic excavation

Here, an excavation task will be decomposed into sub-tasks that are associated with states in UML statecharts, which can overcome the limitations of traditional finite state-machines while retaining benefits of finite modeling (Douglas, 1999). A state is linked in real-time with the machine controllers using information obtained from sensors and estimators to activate the appropriate controllers. Statecharts consist of states, transitions, synch states, and pseudostates. A state includes the existence condition of a subtask and its entry and exit actions. Transitions, triggered by the receipt of events, are executed to accomplish navigation around a statechart. Synch states are state vertices that help model the synchronization of compound tasks or concurrent states. Pseudostates are used as initial, terminal, or transient states that are visited for short periods of time.

The control architecture for our robotic excavator utilizes the capsule types described in the previous section. In particular:

**Expert:** This capsule involves all earthmoving tasks defined for the excavator, and contains the lower-level controllers that are implemented using model-based sliding mode controllers or fuzzy logic controllers. Details of these low-level controllers will be given in Section 5.

**Conflict resolution:** For digging hard soil, a conflict may occur between activated controllers for the boom, such that there is contention between axes for the available hydraulic power and/or pressure. In that case it is necessary to increase the digging force while the boom also needs to be lifted to release hydraulic oil pressure. This capsule is therefore required for properly resolving the conflict.

**Sensor processing:** Window-average filtering is used to calculate the track speeds from the encoders; the capsule also validates operating ranges of the pressure transducers and encoders.

**Safety:** The safety capsule monitors a health message (ON and OK) from all activated object tasks. If any of them fails to send its health message to the safety capsule, then this capsule will set the output vector to 0, closing all hydraulics valves.

In Santos et al. (2000), for the task of digging a trench a task decomposition can be proposed as follows:

\[ \text{Task 1} = \text{Dig a trench of certain dimensions. This excavation task is decomposed into:} \]

- **Subtask 1:** Execute a dig cycle
  - State 1: Position bucket in free space
  - State 2: Digging soil. The digging portion of the work cycle involves the following states:
    - LowerBoom (\( S_1 \)), Penetrate (\( S_2 \)), Drag (\( S_3 \)), and Capture (\( S_4 \)).
  - State 3: Dump
  - State 4: Feasible to continue
  - State 5: Reposition bucket tip

- **Subtask 2:** Obstacle handler
  - State 1: Hard soil handler
  - State 2: Size-particle handler
  - State 3: Hard-surface handler

- **Subtask 3:** Environment tracking (from the laser sensor)
  - State 1: Surface profiler
  - State 2: Bucket tracking

This task can be expressed in our formal syntax as follows:

\[ \text{Task 1} = \text{tuple} \{ \{ \text{Subtask 1, Subtask 2, Subtask 3} \}, \text{priority, entry-conditions, exit-conditions, state-machine}_{\text{main}}, \text{where} \]

\[ \text{priority} = 1, \]

\[ \text{entry-conditions} = \{ \text{TaskSelected, TrenchData, TaskValidated, MemoryAllocated} \} , \]

\[ \text{exit-conditions} = \{ \} \]
exit-conditions = \{\text{TaskFinished, TaskError}\}.
Subtask 1 ::= \text{tuple(functions, priority, entry-conditions, exit-conditions, state-machine}_{op1}), where
functions = \{\text{DoPositionBucketFreeSpace, DoDigSoil, DoDump, DoRepositionBucketTip}\},
priority = 1,
entry-conditions = \{\text{TrenchData, LaserData, InitialPosition}\},
exit-conditions = \{\text{Ready2NextDig, OutOfSurface, SubtaskError}\}.
Subtask 2 ::= \text{tuple(functions, priority, entry-conditions, exit-conditions, state-machine}_{op2}),
Subtask 3 ::= \text{tuple(functions, priority, entry-conditions, exit-conditions, state-machine}_{op3}).

Consider next the task of dragging the bucket teeth along a surface within 5 cm tolerance. Its decomposition is as follows:

Task 2 = Drag a surface for a certain length:
Subtask 1: Execute a drag cycle
  State 1: Position bucket in free space
  State 2: Dig in Soil (only states 3 and 4)
Subtask 2: Obstacle handler
  State 1: Hard soil handler (not used)
  State 2: Size-particle handler
  State 3: Hard-surface handler
Subtask 3: Environment tracking (laser)
  State 1: Surface profiler
  State 2: Bucket tracking (not used).

Note that in general an excavating task can be composed by the subtasks that eventually comprise some common states or element tasks. These element tasks can include “adjust the engine throttle to maintain a constant speed,” “keep the current position for a certain time,” “curl the bucket inward,” “curl the bucket outward,” “rotate the arm inward,” “rotate the arm outward,” “luff the boom up,” “luff the boom down,” “swing the boom to the right,” “swing the boom to the left,” “crawling the tracks forward,” “crawling the tracks backward,” “lift the blade up,” “lower the blade down,” “crawling the tracks right,” and “crawling the tracks left.”

A careful and deliberative task decomposition can allow for a reuse of subtasks. To enhance flexibility, a state-machine that runs only part of its states must have properly defined transitions. Furthermore, modules can also be implemented partially and completed later in an organized fashion. Exception handling uses the implementation of the abstract state-machine that has three states (off, on, and error), inherited by any of the implemented state-machines. Every time the program throws an exception, the exception is handled within that state-machine otherwise it is escalated. In our case, the robotic excavator displays a warning message and stops its operation.

\section{LEVEL CONTROL}

\subsection{Higher-level control}

Hierarchical organization of manipulator control is discussed in Koivo (1989), and behavior-based approaches for robot control are presented in Arkin (1998). A hybrid hierarchical-behavioral robotic architecture, developed by Borrellly et al. (1998), has shown high performance of various autonomous machines. In the work on robotic excavation we propose to combine behavior-based and hierarchical architectures to produce strategies for planning and controlling the machine at higher levels. Practically, excavation tasks can be decomposed into behaviors that activate an appropriate set of suitable controllers. Approximate reasoning with “If-then” rules is incorporated to encapsulate human expert knowledge of earth-moving operations. The description of a particular behavior is based mainly on observation and study of how expert operators command excavators when digging. Using a hierarchical-behavioral approach, a layered control hierarchy of lower and higher levels is proposed for our control architecture. Lower-level controllers are activated upon the technical resolution of conflicts and also of resource sharing, and the management of the control flow and the data flow using statecharts, described in Douglas (1999), and rule-based reasoning. At the higher-level control, the proposed hybrid architecture involves control schemes that are based on decomposition of typical excavation tasks into states or state elements. Each task is represented by a state in a statechart. Unlike finite state-machines as proposed by Fok and Fabuka (1991), statecharts allow for component reuse, concurrency, and for complex nested states (superstates) if required. A statechart is used to represent a state-machine consisting of states and transitions between states, together with synchronization states and pseudostates. Every state object has entry and exit actions, and executes the particular behavior or activity until a transition is set and the state is exited (Santos et al., 2000). Transition conditions can be refined to have event priorities if more than one condition is true at the same time, or can be seen as if/else or switch/case structures. For transition between state elements, a characteristic function \(\gamma_i\) associated with the state \(S_i\) is defined here as follows:

\[
\gamma_i = \begin{cases} 
1, & \text{transition from } S_i \text{ to } S_j \\
0, & S_i \text{ is active, } i = 1, 2, \ldots, r 
\end{cases}
\]

To illustrate this control architecture, consider the tasks of loading, and of digging a trench to a certain depth.
In truck loading, one pass in the task of removing soil from a pile can be decomposed into the sequence of states of positioning the bucket with a specified attack angle, penetrating the soil by curling the bucket inward, lifting and swinging the boom, and then dumping soil by curling the bucket outward. Another common construction task, that of digging a trench as decomposed in Section 4, can be implemented using the statechart shown in Figure 5, where each subtask (state) can call for another level of lower authority as seen in Figure 4. The digging portion of the excavation work cycle and the dump cycle are then considered as substates of this statechart, and will be described in the next section. Transition between task elements is determined mainly by deducing whether the digger has reached a position predetermined according to expert excavator-operator heuristics by measuring the Cartesian position error. Environment sensing is obtained via laser, GPS, or visual sensors mounted on top of the machine (Figure 1) for location and ground profiling to assist with task planning prior to a digging cycle.
5.2 Lower-level control

The usual role of a backhoe excavator in construction is to loosen and remove material from its original location and to transfer it to another location by lowering the bucket, digging by dragging the bucket through the soil, then lifting, slewing, and dumping the bucket load. At the lowest level in the statechart hierarchy are the state elements to be used for the control of machine axes. The actuators driving the boom swing, boom, dipper arm, and bucket attachments of the excavator are axial hydraulic cylinders, and the flow of hydraulic oil to each cylinder is regulated by an electrohydraulic servovalve. Feedback from the arm angular position, hydraulic oil pressure, and actuation force are obtained respectively via absolute encoders, pressure transducers, and disturbance force observers. In moving toward automatic excavation there is a need of accurate position control of the bucket tip. For this, kinematic and dynamic models of excavators are given in Koivo et al. (1996). These models assume that the hydraulic actuators act as infinitely powerful force sources. Instead of tracking desired position or force trajectories, interaction control seeks to regulate the relationship between the end-effector position and the force exerted by the bucket on the soil. Impedance control has been proposed in Ha et al. (2000) as providing a unified approach to both unconstrained and force-constrained motion of an excavator arm. The bucket tip is controlled to track a desired digging trajectory in the presence of environment and system parameter uncertainties. As a result of the impedance control strategy, both the piston position and the ram force of each hydraulic cylinder that are required to exert a given bucket force at a particular position in world space can be determined. The problem is then to find the control voltages that must be applied to the servovalves to track these desired commands. A robust sliding mode control technique is developed to implement impedance control for an excavator using generalized excavator dynamics. The development of these low-level sliding mode controllers, where desired dynamics and robust performance of the closed-loop axis control system are achieved by forcing the system state to remain on a pre-determined sliding surface, has been described in Ha et al. (2001).

In construction automation, there are many situations when the robotic excavator has to operate autonomously to handle hard soil, obstacles or the cases when the bucket becomes stuck. Shi et al. (1995) have shown that knowledge-based control can out-perform model-based control in these situations. Here, in conjunction with the model-based controllers, which are required for fast and accurate tracking of the excavator arm in air and in soft to medium soil, this proposed architecture has the capability of resolving the “hard” soil difficulties by activating knowledge-based controllers using fuzzy logic inference. Associated with the element tasks $\tau_i$ of a typical construction subtask in a statechart that are mentioned in Section 4, fuzzy logic controllers, $FLC_i$, have been developed and can be activated as low-level controllers in conjunction with the sliding mode controllers. For example, $FLC_3$ and $FLC_4$ respectively implement the task elements “curl the bucket inward” ($\tau_3$) and “curl the bucket outward” ($\tau_4$) using the bucket cylinder pressures and the rate of change of bucket joint angle as inputs, and the bucket spool valve opening area as the output. Similarly, $FLC_5$ and $FLC_6$ use the arm cylinder pressures and arm angular speed as inputs and the arm spool valve opening area as the output to implement the element tasks “rotate the arm inward” ($\tau_5$) and “rotate the arm outward” ($\tau_6$) (Santos et al., 2000). The rule sets are, in general, heuristically formulated from observing skillful operators of earthmoving machines. For instance, the element task $\tau_5$ “rotate the bucket inwards” is controlled by $FLC_5$ using the following rule set:

- If (pressure is $PL$ and angular speed is $PL$) then (spool is $PM$)
- If (pressure is $PL$ and angular speed is $PS$) then (spool is $PL$)
- If (pressure is $PL$ and angular speed is $ZR$) then (spool is $PL$)
- If (pressure is $PM$) then (spool is $PL$)
- If (pressure is $PS$) then (spool is $PL$)
- If (pressure is $ZR$) then (spool is $PM$),

where the linguistic labels are defined as $PL = $ positive large, $PM = $ positive medium, $PS = $ positive small, $ZR = $ zero, $NS = $ negative small, $NM = $ negative medium, and $NL = $ negative large, and the pressure, angular speed, and spool position are referred to the bucket axis. The center of gravity method is used for defuzzification. Weights can be incorporated within each fuzzy rule to allow for enabling or disabling, and for further adjusting rule activation if required.

6 FIELD TEST RESULTS

Experiments have been performed using a robotic 1.5-ton mini-excavator. Data acquisition and control algorithms are written in C++ and executed under the Windows NT operating system. The sampling time is chosen to be 10 ms, and data are communicated between the five control system processors by message-passing over a CAN bus at 250 kbit/s. The control hardware organization and machine equipment are reported in Le et al. (1998). Some typical excavation tasks in construction can be executed to demonstrate automation.
In this section, the proposed control architecture for robotic excavation is demonstrated through trench forming, a common task in construction. The average task duration for a single-pass digging portion of the trenching cycle is 15 seconds, which is an average time for a human operator to complete this task. The recorded data for the entry and exit points of the dragging phase are utilized to generate the desired trajectory in joint space for the next digging cycle. Assume that the digging portion of the excavation work cycle is to be executed. The corresponding state “S2-Dig in soil” then involves five main states $S_i$, $i = 1, \ldots, 5$: LowerBoom ($S_1$), Penetrate ($S_2$), Drag ($S_3$), Capture ($S_4$), and LoadToTruck ($S_5$). For illustration, the implementation of state $S_4$ Capture is described here. The entry conditions that the arm is fully contracted or the bucket is full or the bucket tip is out of the soil stated in the transition for this state must be true. Then state $S_4$ will be executed by running the associated lower-level controller. If the bucket is horizontal in the case of full bucket filling, then the transition condition for $S_5$ is true and the next state LoadToTruck will be executed. Additionally, a time-out transition is implemented so that if the bucket gets stuck or cannot finish the dig in an allocated time, the rule base will activate knowledge-based controllers or provide a new strategy to remedy the situation. This strategy could be discarding the contents if the captured soil volume is less than 50% or else going to state $S_5$. Field tests have been conducted that involve trenching in soils categorized as “soft,” “medium,” and “hard.” The recorded data for the entry and exit points of the dragging phase are employed to generate the desired trajectory in joint space for the next digging cycle. Figures 6–8, show the measured Cartesian trajectory of the bucket tip corresponding to each phase for these categories, respectively. In the figure, segments AB, BC, CD, and DE correspond respectively to the previously defined phases $S_1$, $S_2$, $S_3$, and $S_4$. In the absence of hard inclusions, the tip motion during the dragging phase is observed in Figures 6 and 7 to satisfy a desired tolerance of 5 cm. Figure 8 shows that the bucket teeth cannot, however, penetrate “hard” soil smoothly because the required soil cutting force exceeds the excavator’s force capacity. The bucket must then be lifted, and the surface scratched with the bucket teeth to loosen the soil underneath.

Figures 9–11 show the hydraulic pressures measured at the head side (solid line) and rod side (dashed line) of the bucket (top), arm (middle), and boom (bottom) cylinders for the cases of digging soft, medium, and hard soil, respectively. The force generated by each cylinder
Fig. 7. Bucket tip trajectory, digging “medium” soil.

Fig. 8. Bucket tip trajectory, digging “hard” soil.
Fig. 9. Hydraulic cylinder pressure measurements during trenching “soft” soil.

Fig. 10. Hydraulic cylinder pressure measurements during trenching “medium” soil.
Fig. 11. Hydraulic cylinder pressure measurements during trenching “hard” soil.

can be estimated approximately from the pressure difference across the cylinder. It is observed that digging medium and hard soil requires large forces, as one would expect. The marked differences, observed in the oil pressure measurements of Figures 9–11 when trenching soil with various stiffness, correspond to only slight variations in the bucket tip trajectories for three different kinds of soil.

7 CONCLUSION

In this article we have proposed a control architecture for robotic excavation with the construction automation focus. A formal and graphical architecture framework based on UML statecharts and capsules has been developed and applied to the autonomous execution of some excavation tasks of a hydraulically actuated robotic excavator. The control architecture is designed with a view to managing hierarchical complexity, and facilitating the application of formal verification methods, software reuse, and lower-level control results. For higher-level control, behavioral and hierarchical approaches are combined for decomposition and execution of some excavation tasks.

At the lower level, fuzzy logic and sliding mode control are used in conjunction to achieve robustness of the closed-loop axis control system and to capture expert experience when handling unmodeled situations encountered in digging hard soils. Field tests of digging various types of soil show promising results. At this stage relatively hard soil and some included hard obstacles can successfully be handled. It is believed that the methodology can be extended to coordinated control of complicated autonomous machines at many scales and with a variety of distinct dynamic operating regimes. The experimental results described here suggest the technical feasibility of achieving autonomous robotic excavation in moving toward construction automation.

ACKNOWLEDGMENTS

This work is supported, in part, by the ARC Centre of Excellence programme, funded by the Australian Research Council (ARC) and the New South Wales State Goverment. Assistance during field tests provided by Dr. Q. H. Nguyen and M. Santos is gratefully acknowledged.
REFERENCES


