Chapter 1

Robots that Learn?

Robots' usefulness is generally limited to simple repetitive chores in never–changing, error–free environments, such as assembly lines for automobile manufacture and canning establishments. Yet robots were designed to be flexible, programmable, general-purpose machines; "the key characteristic of robots is versatility" [Lozano-Perez 1983]. Indeed, robots have often been proposed as autonomous entities which possess human intelligence and operate in our real world [Capek 1923, Moravec 1988] — a step still far from reality. Even people still experience difficulty in using them.

Machine learning is generally limited to small, synthetic domains, leaving many people skeptical of its usefulness [Parnas 1987]. Although some learning systems have been applied to controlled classification problems,[1] machine learning would greatly benefit from further successes in the real world. One researcher goes so far as to say that machine learning needs a backbone, namely, a robot body interacting with our real world [Andreae 1988].

The ambition of this thesis is to exploit machine learning to obtain robots which are more useable by the general population. A natural interface for people is demonstration — designating tasks by showing. This thesis presents the ETAR system which has been implemented on an Excalibur robot. ETAR stands for *Example-based Task Acquisition in Robots* and it accepts examples in which the user directly leads the robot. It generates a procedure with loops, conditionals, and variables, allowing task completion in a variety of circumstances. In the past, robotics and machine

[1] *E.g.* ID3 [Quinlan 1986] to contact lense prescriptions and A^q [Michalski & Chilausky 1980] to soybean disease analysis.

learning have remained separate [Segre 1988] but ETAR enforces their unity.[2] It is a significant contribution to machine learning because the implementation is on a real robot, both qualitative and quantitative reasoning methods are employed, and issues about activities in the external world versus symbol classification are prominent. This thesis is also a significant contribution to robotics, since it stresses an alternative to robot programming and is a step towards capable robots in our dynamic world.

This chapter summarizes the current robot task specification methods to introduce some robotics terminology, but mainly to emphasize the difficulties and limitations in present day robot interaction. To alleviate these problems, machine learning by example is proposed as a necessity in robot paradigms. The objectives of this work are stated, and a summary of the remainder of the thesis concludes the chapter.

1.1 Specifying a Robot Task

Robots have remained unsophisticated because of the difficulty humans have in expressing tasks to them. These difficulties become apparent when investigating how users currently interact with robots. To classify task stipulation methods, different authors use different terminology but five levels are predominant [Aken & Brussel 1988, Bonner & Shin 1982, Gini & Gini 1985, Lozano-Perez 1983, Rock 1988]. Figure 1.1 illustrates the hierarchical relationship between these levels, which are now discussed beginning from the bottom.

At the *joint level*[3] a task is defined as direct control over the robot's joints, usually through the robot controller. A task description is often a sequence of numbers, *e.g. joint angles* (also called *joint coordinates*) where the robot position is

[2] Just as ARMS [Segre 1988] does.

[3] Some authors [Bonner & Shin 1982, Rock 1988] further divide this level in two: microcomputer/hardware level and point–to–point level. Reasons for this become clear in this paragraph.

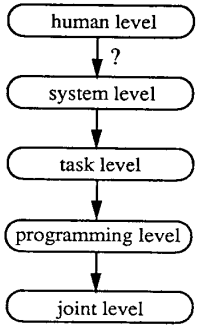


Figure 1.1: Methods of specifying robot tasks

described as the angle of each link with respect to the previous link. Explicitly coding tasks in this manner is impractical. Instead, easily implemented *teach by guiding* or *direct lead* mechanisms allow the user to physically move the robot through the task, while manipulator joint positions are stored, and repeated later in precisely the same order. This interface is valuable since the user has visual control over the task performed. However, there are problems. Tasks containing loops require explicit movement of the robot through each of the iterations and the number of iterations is then fixed. Conditional branches are impossible and environmental changes necessitate teaching the entire task again. Tasks requiring feedback and compliance are unlikely candidates for this approach. Questions concerning real world uncertainties remain unanswered and positioning the manipulator to achieve the required accuracy is often a chore for the user.

At the *programming level*, users must program tasks in a robot control language. These languages contain motion primitives, and sometimes higher level control structures,[4] to move the robot end effector in *Cartesian* or *world coordinates*. In world coordinates, the user manipulates a location (x, y, z) and an orientation (often

[4]Hence [Bonner & Shin 1982] and [Rock 1988] break this level in two: the primitive motion level and the structured programming level. [Lozano-Perez 1983] calls this the robot level.


```
     FOR I := 1 TO 4 DO
     BEGIN
          ARM ::= - BO + Z + T6        ; work wrt to bolt
R7:       MOV    BG + A               ; approach bolt
R8:       MOV    BG                   ; move down on bolt
          GRA                         ; grasp bolt
          TOL ::= E - BA + BT          ; control bolt tip
                  ⋮
```

Figure 1.2: Example of the PAL programming language (adapted from [Takase *et al* 1981])

roll, *pitch*, *yaw*). See [Bonner & Shin 1982], [Gini & Gini 1985], and [Lozano-Perez 1983] for examples and comparisons of the numerous robot programming languages available. Figure 1.2 is an example of one of these languages. It shows part of a task specified as a PAL program [Takase *et al* 1981], with assignment (::=), relative coordinate frames for objects (BO, T6, BG, ...), matrix multiplication (+) and inversion (−), and control structures (FOR). Tasks are translated into joint level commands before being sent to the robot. Using this level requires vast expertise, since the user is responsible for defining objects as positions with relative coordinate frames in three–space, ensuring that the path travelled by the robot is predictable and collision–free, integrating sensory information, controlling speed and acceleration of the robot's joints, building smooth transitions in the robot motions, and accounting for dynamics. An eventual goal for programming level systems is to handle these problems implicitly, but for now, even the best contains only a subset of them [Aken & Brussel 1988].

The *task level* gives the user a subset of natural language with which to describe robot tasks. A task is expressed as a relation between objects, *e.g.* "PLACE board2 ON TOP OF board1". These commands are translated into the lower levels by a built–in planner which is responsible for generating manipulator positions and collision–free paths, and for determining stable grasp positions. Task level languages, such as RAPT [Ambler *et al* 1982, Poppleston *et al* 1980], are still experimental and

are limited by the development of path planners and successful implementation of the programming level. Once available, task level languages will be advantageous since tasks are specified without requiring awareness of the mathematics in low level manipulation and geometry. However, the user must still provide the sequence of sub-goals to manipulate the objects into the desired final position.

Task descriptions at the *system level*[5] are also specified as a subset of natural language. Now the user indicates the desired goal, *e.g.* "make a boat". Plausibility of systems at this level awaits further advances in artificial intelligence planners, natural language interfaces, reasoning systems, and the problems arising in the lower levels of task specification. [Rock 1988] speculates that this level is composed of two parts — one responsible for planning and the other responsible for coordinating parts of the task and environmental features such as conveyor belts and other robots. He indicates that expert system control and object-oriented approaches will be required.

1.1.1 Summary: A problem in task specification

The only *widely available* methods for designating tasks to robots are at the joint and (partial) programming levels. This section has shown that unless the user is a robot programming expert, these levels are impractical for specifying a general robot task. Furthermore, there are two problems at the *proposed* task and system levels:

1. unaccounted tasks or conditions. System level proposals incorporate intelligence, but only with planning and deductive inference methods — no capacity to acquire entirely new knowledge is suggested. Thus, unless the system designers have accounted for a task, it cannot be done.

2. specification must be in (a subset of) natural language. Users of the task level must be able to decompose the task into the proper subtasks and then express them in appropriate natural language phrases; users of the system level need only find the one nonambiguous phrase for the task. But people do not always mean exactly what they say, and often resort to demonstration to back up their verbal explanation.

[5] Called the objective level in [Aken & Brussel 1988].

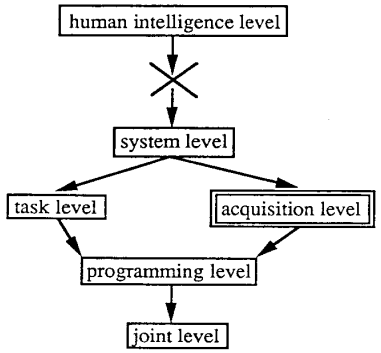


Figure 1.3: Acquisition level added to methods of task specification

[Rock 1988] says that the system level is the link to *human intelligence*, however these problems break that link. Instead, the user must again resort to robotics programming expertise to solve them. The ultimate goal of this thesis work is to make robots so easy to use that almost everyone could. Another level is required.

1.2 Acquisition Level

A new level, the *acquisition level* where robots obtain knowledge during operation, must be added to the robot paradigm, as shown in figure 1.3. This thesis concentrates on one aspect of the acquisition level — learning tasks from humans.[6] This is because the use of robots must be simplified so that any person, knowing how to successfully complete a task, may delegate it to a robot. Were this transfer of knowledge to occur between two people, the knower would take on the role of a "teacher", both

[6] Versus, learning in changing environmental conditions, *etc.*

showing and telling the method. Extending this to machines — while retaining a similar interaction for the human teacher — implies that a robot must learn from the examples and instructions of the user. This work focuses on learning from the examples of the user.

1.3 The Goal of the Thesis

The need for an acquisition level has been established for robots that are to become more useable. Learning from user examples constitutes an important part of this level which this thesis investigates through the synthesis of a prototypical learning system. This is the goal — *to design and implement a system which acquires robot tasks from examples.* The system must satisfy the following objectives:

1. It operates together with a *real* robot. Many systems have been implemented in simulated domains (*e.g.* [Andreae 1984, Kadie 1988]), but if a system is to aid a user in a real world task then it must work on the actual robot. This means that learning begins at the joint level, conversion to and from world coordinates must be done, motion planning is an essential part in the execution of a task, and actual robot actions are learned in the dimensions of the joint space.[7]

2. The examples are obtained from a direct lead mechanism. Direct leading is the best interface for task specification for three reasons:
   (a) the user controls the machine to the desired positions and is able to see the outcome.
   (b) direct lead mechanisms are widely available.
   (c) natural language interfaces are still experimental and incomplete. Furthermore, direct lead mechanisms avoid the ambiguity of natural language.

[7] 7 dimensions in the Excalibur robot.

   Some systems jump into learning at a higher level — each example is a trace which decomposes the task into symbolic motions and arguments (*e.g.* [Andreae 1984], [Segre 1988]). This trace is constructed by the user, probably with difficulty. It must be possible to acquire a task directly from the robot feedback.

3. Examples are provided by a user who is not expected to have knowledge of the underlying learning method. The motivation of this work is to make robots easier to use. Thus, if a user can do the task, then that is sufficient for teaching. The user must never be expected to understand the acquisition system first.

4. Reasonable examples may be expected. This objective is not a contradiction to the previous one. Rather, every learning system (including humans) requires felicity conditions [VanLehn 1983] in order to succeed. This may be as simple as ensuring that the user does not perform useless subtasks in the teaching of a task.

5. The system requires minimal domain knowledge, which would be replaced by perception equipment if it was available. Again, to make tasks easier for people, a user must not be required to initialize system background knowledge.

6. The system is an inductive learning system. It must be able to acquire new tasks, rather than just learn extensions or different configurations of previously known tasks (as in deductive learning).

7. The system can handle loops, conditionals, and variables. If general tasks are to be learned and repeated in a variety of circumstances, then control structures and variables must be an explicit part of their description.

Before such a learning system can be implemented, several preliminary objectives must be fulfilled:

1. Investigate machine learning systems to isolate the types of knowledge they learn and determine how their methodology may be applied to robotics.

2. Implement a programming level adequate enough to support an acquisition level. Two fundamental routines must be included:
   - kinematics — a conversion between joint and world coordinates.
   - motion interpolation — controlled motions for moving the robot on a straight line path and for changing its orientation smoothly. These are essential in the execution of any robot task.

   The Excalibur only had a joint angle controller, and even that had errors in it.

1.4 Description of Remaining Chapters

This work develops and implements an acquisition system, ETAR, which learns robot assembly tasks from examples. Sample tasks include stacking objects, placing objects on conveyor belts, and constructing widgets. The remainder of this thesis details ETAR. It indicates ETAR's relation to past work, provides examples of a robot acquiring tasks, evaluates the system, and indicates further directions which robot learning systems must take.

Chapter two indicates the capabilities of the proposed system through an example of a task. It describes the domain of this application and demonstrates each phase of the learning algorithm as ETAR acquires a procedure for this task. The result of this chapter is an intuitive feel for ETAR. The example is the basis for the detailed description of the learning system to follow.

This thesis stems from many excellent works in five major areas: traditional concept learning, function induction, automatic program construction, human skills acquisition, and task learning in simulated and real robot domains. Chapter three summarizes major systems in each of these areas and indicates their relation to ETAR.

Chapter four elaborates the learning algorithm. Each of the main components — knowledge representation, symbol processing, outer generalization, inner

induction — is detailed as a section of this chapter.

The underlying aspects of the Excalibur robot used in ETAR's implementation are described in chapter five. The chapter indicates the low level capabilities on top of which ETAR is built. Forward and inverse kinematics are provided. A little used quaternion–based interpolation method is discussed as it applies in the implementation of the primitive robot motion functions of which tasks are composed.

ETAR is almost completely implemented and chapter six indicates the extent of this and proposes directions for future work. The chapter further analyzes the assumptions underlying each of the learning components, and then combines them to generate a formal grammar for the system.

Chapter seven concludes the thesis and summarizes all that is accomplished.

1.5 Abbreviations Used in Thesis

Three abbreviations are used throughout the thesis:

ETAR Example–based Task Acquisition in Robots
FOA Focus Of Attention
PMF Primitive Motion Function.

Chapter 2

Scenario

This chapter introduces ETAR. The goal is to provide insight about its capabilities and how it works without the worry of extensive detail. Hence, this chapter illustrates a typical task learning example, briefly showing the progression through the ETAR algorithm. Later, in chapter 4, this example recurs as each part of the system is detailed.

ETAR learns simple assembly tasks from the user guiding the robot. As examples are observed, ETAR samples the robot's joints, eliminates extraneous feedback, models each example with a set of robot programming primitives, searches for loops and branches, and merges the examples to result in a general task procedure. This procedure is added to the knowledge base where it can be accessed for future jobs.

2.1 Preliminaries to Learning

A classic task, together with background information about the robot and the task, introduces the ETAR learning episode. First this section explains the task domain, specification, and restrictions and presents the examples. The teaching method and the robot of ETAR's implementation are described next. Finally the prerequisite knowledge is stated.

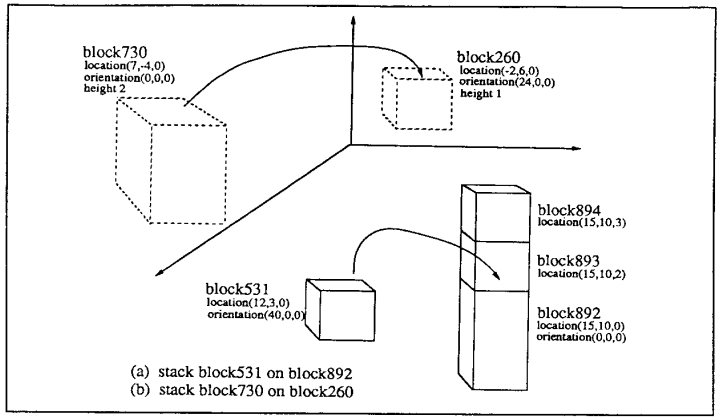


Figure 2.1: Examples of a task

2.1.1 The Task

ETAR works alongside a robot which is manipulating a real three–dimensional blocks world.[1] When a task is unknown it is ETAR's responsibility to learn it. A task ranges from stacking objects to placing objects on conveyor belts to constructing widgets and is currently specified as

$$aTask(parameter1,\ parameter2,\ parameter3, \ldots).$$

For example, if the robot is to stack block531 on top of block892, as in figure 2.1, this is stack(block531, block892) where block531 and block892 are *explicit parameters* to the task. When stacking is unknown, ETAR must learn the general task stack(x, y) where x and y are arbitrary objects.

Learning occurs from examples. Two examples of the stacking task are depicted in figure 2.1. In (a) the robot must remove two extraneous blocks before stacking

[1] The domain can vary, provided it can be represented in the background knowledge. More discussion is in chapter 4.

the desired ones while in (b) it simply places one block on top of the other.[2] Now attentions focusses on how the robot obtains these examples.

2.1.2 The Examples: Leading the Robot

ETAR is implemented with an Excalibur robot [Excalibur 1986]. This robot consists of a slave (often referred to as *the robot*) and a master manipulator. The slave is an arm with six revolute joints and a gripper that opens or closes two fingers. Further geometric details of the robot are in chapter 5. The master is a small model coupled with the slave to enable user control.

The easiest way for a user to provide examples is by *leading* the robot through the task. This is accomplished by physically moving the master arm so that the robot follows to complete the task in the actual workspace. During leading, the robot's joints[3] are sampled to yield a *teach path* or *sequence*. Teach paths are standard in robot systems and are usually played back to execute a never–changing task. Using the same feedback, ETAR discovers a general procedure with variables, loops, and conditionals.

To make learning feasible, ETAR has four main assumptions at the leading level:

- The user has the ability to perform the task with the robot.
- At any time, the robot manipulates a maximum of one object.
- Leading is consistent, *e.g.* if the user picks up one object by approaching from the top then all objects will be picked up from the top and not by side approach, unless there is a reason.
- Obstacle avoidance is a separate problem. The current ETAR implementation contains none. Hence, the paths of the robot links are never considered. The

[2]The order of these examples is not important.
[3]Actually the master arm joints, but they correspond to the robot's joints.

position and orientation of the gripper is important, but any gripper obstacle avoidance must be carefully taught.

2.1.3 Background Knowledge

How can a system learn from a sequence of many joint angles? In ETAR learning is aided with its background knowledge, of which there are three kinds. First, a *workspace model* denotes the objects in the manipulator environment. Its main content is information normally available from perception systems and in computer-aided design, such as the positions and dimensions of objects. The other prerequisite knowledge is a set of *primitive motion functions* (PMFs) or types of motions into which all robot tasks can be decomposed, and a set of *numerical basis functions* used in the later stages of the inductive process.

2.2 Learning the Task

With preliminaries aside, task learning commences. When the robot is commanded to stack(*block*531, *block*892), it searches its knowledge base for the procedure. Since stack is unknown, the user must teach the robot by leading examples of the task. ETAR observes each example as a hexadecimal teach sequence. Utilizing the examples in figure 2.1, this section shows how these teach sequences generalize to a stack procedure. First each example converts to a symbolic form, and then primary generalization begins. During primary generalization each example is matched within itself to isolate loops, matched with other examples to yield the common parts and differences of a task, and finally merged into a procedure as variables and conditions are acquired.

2.2.1 Conversion to Symbolic World Motions

Many points are collected for each lead example. A two minute task, sampled at 10 *Hz*, yields a teach sequence of 1200 joint positions such as those at the top of figure 2.2. ETAR condenses the sequence by eliminating points which are not essential in the task. A point is kept if the robot gripper is near one or more objects. These objects are the *focus of attention* (FOA) and play a main role throughout the learning algorithm. The FOA is the upshot of scanning the workspace for objects around the gripper. In ETAR's case, the objects' positions are part of the background knowledge, expressed in world coordinates. Hence, the joint teach sequence is transformed into world coordinates and every point near an object is augmented with its FOA and remains in the teach sequence. Other points are discarded.

Partitioning the sequence into PMFs is the final step of symbol processing. Straight line motion *translate*(x, y, z), smooth orientation change *rotate*(*roll*, *pitch*, *yaw*), orientation change simultaneous with straight line motion *transRot*(x, y, z, *roll*, *pitch*, *yaw*), gripper opening or closing *grip*(*open*/*close*), and path independent movement *moveto*(x, y, z, *roll*, *pitch*, *yaw*) are a sufficient set of PMFs into which all Excalibur tasks may be decomposed. World coordinate teach paths are divided into subsequences, each of which is replaced by the appropriate PMF affixed with its arguments. The FOAs always remain with the sequence. Figure 2.2 portrays the output at each stage of symbol processing. Depicted in figure 2.3 is the stacking example once symbol processing is complete. The double-headed arrows result from primary generalization — the topic of the next section.

2.2.2 Primary Generalization

Once the sequence is in symbolic form, the main generalization occurs at two levels:

- *outer* — matches the PMFs within and between examples, *e.g. translate* to *translate*

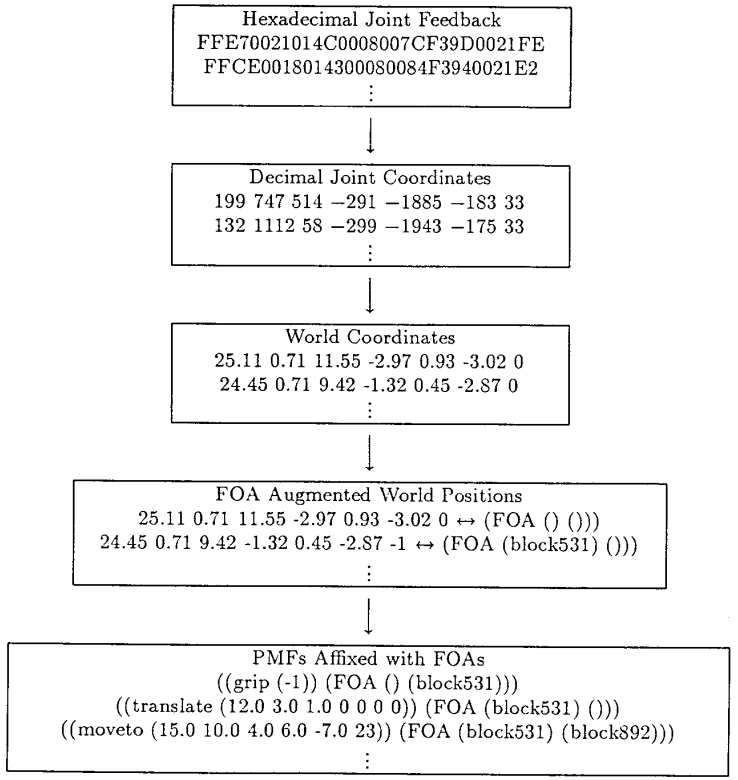


Figure 2.2: Symbol processing

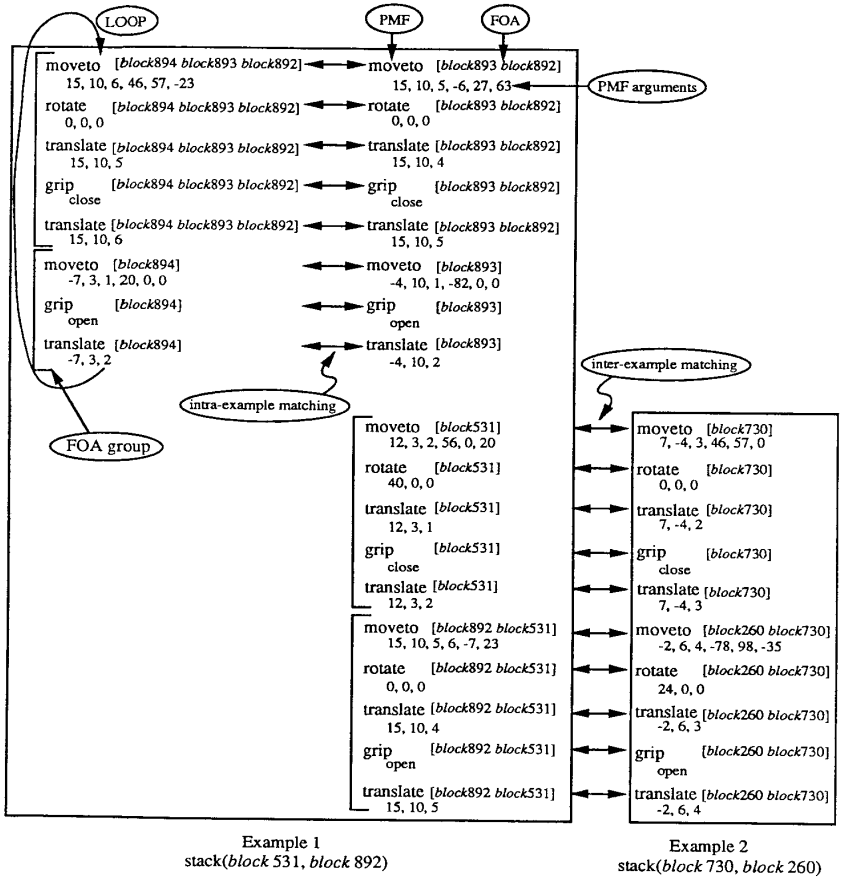


Figure 2.3: The stacking example after symbol processing and outer generalization

- *inner* — merges examples by inducing arguments to the matched PMFs, *e.g.* general x, y, and z parameters of *translate*, and by finding conditions for loops and branches.

Outer Generalization

Intra–example processing, the first step in outer generalization, standardizes each example. This is achieved by grouping the sequence according to FOA, combining some PMFs with the use of production rules to account for user overshoot and correction, and searching for loops. Outer generalization ends with *inter*–example processing. Here similarities in the PMFs and FOAs between examples are matched, whereas differences form a branch. Figure 2.3 exhibits the FOA groups, a loop formed, and the matches determined by outer generalization.

Inner Induction

Inner induction merges the examples to complete ETAR's task acquisition process. First, the PMFs which outer generalization has matched are coalesced into one PMF with general arguments. The PMF arguments are obtained by FOA–controlled function induction on the composition space of the basis functions. Finally conditionals to the loops and branches in the examples are generated. Qualitative discrimination finds differences in the knowledge base during and after a loop, and before branches. These differences become the loop termination or branch conditions.

stack block1 onto block2
  UNTIL (block2 under nothing) *remove any blocks from* block2
    foa$_1$ = highest object on top of block2
      approach and pick up top block
    moveto(15, 10, $z_{foa1} + height_{foa1} + approachDist_{robot}$, ?, ?, ?)
    rotate(0, 0, 0)
    translate(15, 10, $z_{foa1} + height_{foa1}$)
    grip(*close*)
* translate(15, 10, $z_{foa1} + height_{foa1} + approachDist_{robot}$)
* moveto(?, ?, 1, ?, 0, 0) *move it away*
    grip(*open*)
    translate(?, ?, 2)

  foa$_1$ = block1 *pick up* block1
  moveto(x_{block1}, y_{block1}, $height_{block1} + approachDist_{robot}$, ?, ?, ?)
  rotate($roll_{block1}$, 0, 0)
  translate(x_{block1}, y_{block1}, $height_{block1}$)
  grip(*close*)
* translate(x_{block1}, y_{block1}, $height_{block1} + approachDist_{robot}$)

  foa$_{1,2}$ = block2, block1 *place* block1 *on top of* block2
* moveto(x_{block2}, y_{block2}, $height_{block1} + height_{block2} + approachDist_{robot}$, ?, ?, ?)
* rotate($roll_{block2}$, 0, 0)
* translate(x_{block2}, y_{block2}, $height_{block1} + height_{block2}$)
  grip(*open*)
  translate(x_{block2}, y_{block2}, $height_{block1} + height_{block2} + approachDist_{robot}$)

* indicates an updated object position

? indicates a "don't care" position parameter

Figure 2.4: The stacking task learned

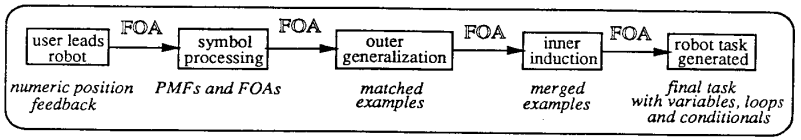


Figure 2.5: The ETAR system

2.2.3 Task Acquired

The stacking task learned by ETAR is shown in figure 2.4. Through this figure the system capabilities are highlighted and the algorithm is summarized. ETAR analyses typical feedback, which is available in most robots' teach by guiding mechanisms. Both quantitative and qualitative investigation guide the system to a general procedure which:

1. may contain variables to characterize the objects being manipulated, *e.g.* the first "moveto" of the figure depends on the object's x and y position.

2. may contain functions of the objects' characteristics, *e.g.* the first "moveto" also depends on a function (+) of the object's z position, its *height* and the robot's *approachDist*.

3. shifts the robot's attention to the objects it should manipulate by using an FOA mechanism, *e.g.* in the figure, the attention changes between block1 and block2 and any extra objects.

4. may contain loops with a qualitative termination condition, *e.g.* the first step in the task of the figure is a loop with an UNTIL condition. Branches may also be present and are designated by similar conditions.

5. may contain arguments whose value does not matter, *e.g.* the '?'s in the figure.

Figure 2.5 summarizes the ETAR method. First, every example passes through a symbolic processing state to condense the numeric robot feedback into a sequence of PMFs and FOAs. Primary generalization (outer and inner) takes this sequence and

forms the final procedure by merging the examples after matching the PMFs and inducing arguments to them. The FOA is the principal control at each step of the learning process.

Chapter 3

Learning is Not New

People continue to learn all their lives, yet defining learning or determining how it occurs is difficult. Learning theory has long been a forefront of research, primarily in the areas of psychology and education. More recently, other fields such as Computer Science, through Artificial Intelligence and Machine Learning, have proposed, implemented, and simulated knowledge acquisition systems in machines [Cohen & Feigenbaum 1982, Dietterich & Michalski 1986, Michalski *et al* 1986, Mitchell *et al* 1986a]. This chapter provides an overview of these systems, concentrating on those which, like this thesis, propose to learn from examples.

To compare ETAR with other learning systems, two divisions for knowledge are helpful — open versus closed and static versus active. The first section of this chapter clarifies these distinctions. Section two discusses learning by example, primarily through summaries of the diverse computer learning systems around today. Here it is apparent that acquisition of open, active knowledge, necessary in robotics, is ignored by all these systems. Section three turns to psychology and physical education for answers in the area of human skill acquisition. Three Machine Learning systems, ARMS, DIFFY-S, and NODDY, focus on robot learning issues. They provide the motivation for this thesis and are examined at the end of the chapter.

3.1 Types of Knowledge

Two classifications for knowledge provide the basis for understanding how ETAR fits in a framework with other example-based learning systems. Both classifications are not well-defined and have always been a source of trouble for Machine Learning

researchers.[1] But even so, these categories highlight areas missing from current Machine Learning research, yet necessary in robotic applications. The first distinction is at the *interaction* level [Holding 1981]:

1. *open* — references external elements in the environment, *e.g.*, how to hit that ping-pong ball, that ice-cream cone (specific).
2. *closed* — does not reference external elements in the environment, *e.g.*, how to add two numbers in the head, an ice-cream cone (general).

Another distinction occurs at the *usage* level, *i.e.*, how to use the knowledge — a hard categorization to make. Clearly, knowing the shape of an object is different from the ability to play ping-pong. This distinction may be captured as follows:

1. *passive* — Carbonell, Michalski, and Mitchell [Carbonell *et al* 1986] simply call this "knowledge". They define it through an example of physics "knowledge" as "significant concepts of physics", "their meanings", and "their relationship to each other and the physical world". "Knowledge" includes

   > descriptions and models of physical systems and their behaviors, incorporating a variety of representations — from simple intuitive mental models, examples and images, to completely tested mathematical equations and physical laws. [Carbonell *et al* 1986]

   Knowledge of this type often embodies facts, hence this thesis refers to it as passive knowledge and adopts the above definition. Though this knowledge is often represented *declaratively*

   > in which most of the knowledge is represented as a static collection of facts accompanied by a small set of general procedures for manipulating them [Rich 1983],

   the development of functional and logic programming languages leaves the distinction between facts and procedures unclear. Hence, this classification is not concerned with representational categories, just about the contents of the knowledge. Examples of passive knowledge include a general ice-cream cone, that specific ice-cream cone, and $\sin^2 x + \cos^2 x = 1$.

[1] The great procedural/declarative debate is still going [Rich 1983].

2. *active* — This type of knowledge includes what many people call physical and mental *skills* [Carbonell *et al* 1986, Schmidt 1982]. Active knowledge expresses *how* to do something, is always represented *procedurally*,[2] and implicates space and time. Some examples are hitting a ping–pong ball and adding two numbers in the head.

This thesis is concerned with open, active knowledge, *i.e.*, knowledge to manipulate an environment which contains entities other than the manipulator. Such knowledge will be called *task knowledge*. Passive knowledge must be present in the system before any tasks can be learned. Consider block stacking. Before this task can be done, the system must know about "block". ETAR assumes a store of such knowledge (which could have been acquired by traditional concept learning systems, in computer–aided design, or from other sources).

3.2 Learning from Examples

There are many aspects to learning, but the best understood method is learning by example [Cohen & Feigenbaum 1982]. From a teacher's perspective, using examples is a vital technique for presenting new concepts, especially active knowledge:

> To instill in the learner the nature of the goal of the act together with the action(s) that will accomplish the desired outcome, instructors inevitably resort to demonstration to supplement instructions. [Newell 1981]

Demonstration benefits teaching for at least two reasons:

1. The teacher often experiences difficulty in verbalizing the concept. It is particularly hard to describe "how" to do something.

2. The learner obtains a visual or mental picture, while avoiding the ambiguity of a natural language description.

[2] In the sense of [Rich 1983].

These two reasons apply to robotics as well. Tasks are hard to program and using a natural language interface is beyond the capabilities of current technology.

Machine Learning systems that learn from examples are abundant. This section provides an overview of some. The goal is to recognize the type of knowledge Machine Learning systems typically handle, and that this does not include task knowledge.[3] However, ideas from these Machine Learning systems are expanded and incorporated into ETAR.

3.2.1 Traditional Concept Learning

Concept learning systems use examples of a concept to construct a general description. One system is set apart from another primarily by the underlying mechanism used to guide the search for identification rules.

Version Space — classification in hierarchical domains

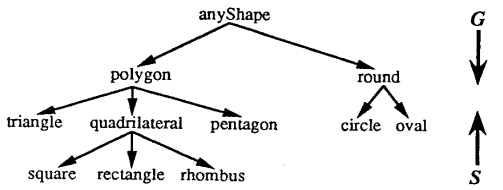
Version space [Mitchell 1982, Mitchell 1977] employs representation languages in which the sentences form a partial order according to generality. Any concept is bounded by two sets:

- S: the maximally specific set, containing those descriptions which definitely fit the concept. Initially, S contains the starting positive example of the concept, and further positive examples force S to be generalized by climbing up the hierarchy.

- G: the maximally general set, containing a description which does not include any negative example encountered so far. G is initially the most general element in the language since no constraints have been imposed upon the concept.

[3] A later section focuses on Segre's ARMS [Segre 1988] which acquires task knowledge. ARMS is not a "typical" Machine Learning system.

Learn "polygon", given this hierarchy:



step	example type	example	G and S sets
1	positive	square	G = {anyShape} S = {square}
2	positive	triangle	G = {anyShape} S = {polygon}
3	negative	circle	G = {polygon, oval} S = {polygon}
4	negative	oval	G = {polygon} S = {polygon}

Figure 3.1: Version space example

Negative examples cause G to be specialized by moving down the knowledge hierarchy.

If $S = G$, the concept has been isolated and learning is complete. Version space is illustrated in figure 3.1. The goal is to learn the concept "polygon" from the given geometric hierarchy. G is initialized to the top element of the hierarchy, and S becomes the first positive example (step 1). Since the second example is positive, G remains fixed in step 2, but S must be generalized to include the example. S becomes {polygon}, the minimal element in the hierarchy which covers the new example and the current S. The third example is negative so now G is specialized to exclude the example while S does not change. One last negative example finishes the learning as $G = S =$ {polygon}.

At first glance, version space seems useless since it does not gain any information not already present in the hierarchy. Its worth comes on incorporating version space into other programs, such as LEX [Mitchell *et al* 1981] which learns categories for

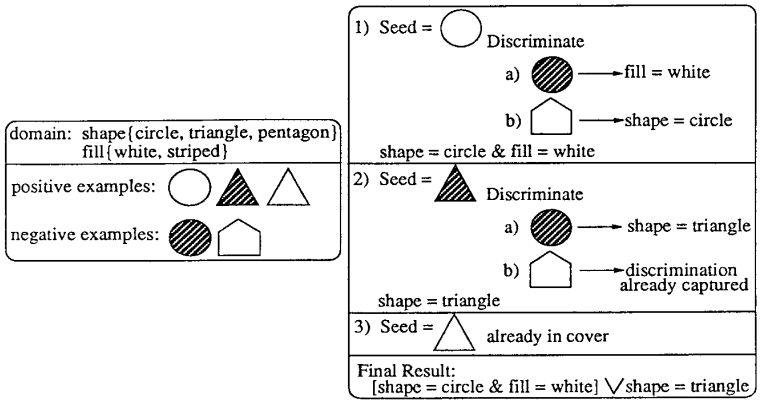


Figure 3.2: A^q example

applying integration rules. In generalizing FOAs, to determine which workspace characteristics are relevant to the task, ETAR climbs the object hierarchy just as in the version space S set generalization.

A^q — qualitative discrimination

A^q [Falkenhain & Michalski 1986, Michalski & Chilausky 1980] is passed a complete set of positive and negative examples and it must learn a description distinguishing the positive ones from the others. The language is lists of attribute–value pairs. Beginning with a positive example, called the *seed*, A^q finds a minimal conjunction of these attribute–value pairs which discriminates the seed from all negative examples. Disjunctions are added when new seeds are chosen and sent into the discrimination process.

A simple example is in figure 3.2. With two attributes, shape and fill, the white circle and the white and striped triangles must be isolated from the striped circle and the pentagon. How is the first positive example (the white circle) different from

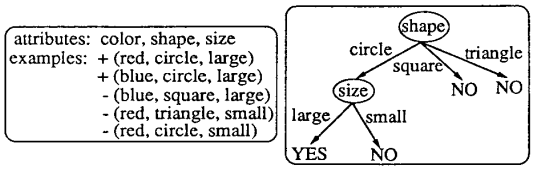


Figure 3.3: ID3 classification example

the negative examples? A^q looks at each negative example, and if it is not already eliminated by previous conditions, discovers a discrimination condition to add to the concept description. The striped circle differs from the seed by the fill value and the pentagon differs in shape. These conditions become the concept description, to which more disjunctions are added until the entire positive set is accounted for.

The attribute–value pairs of ETAR's frames are well–suited for the A^q algorithm. ETAR uses it for qualitative discrimination in determining the conditions for the loops and branches. Incorporation of the FOA enables ETAR to quickly discover the conditionals.

ID3 — branching on explicit information provided in examples

ID3 [Quinlan 1986, Quinlan 1983] classifies objects, represented as attribute vectors, by discovering a decision tree from a complete set of examples. Each example has one attribute which is the classification attribute and usually indicates whether the example is positive or negative. The training set is partitioned on the basis of one attribute, which is not the classification attribute. This becomes the root, its values form the branches, and the examples corresponding to each value are further divided until all examples in the branch have the same classification attribute value. This value becomes the leaf of the branch. Branching is determined by information theoretic measures so the chosen attribute results in the largest gain of information. Figure 3.3 contains a simple example of ID3. Since the shape attribute has the highest entrophy, it is the first branch. Size is next, and color is irrelevant since it

contains no information because both colors are present in the positive examples. The generated decision tree can be converted into production rules. Other methods, such as PRISM [Cendrowska 1988] use a similar strategy to produce these rules directly and more efficiently. Each production is an "if–then", resulting in a program which determines whether objects are members of the learned concept.

Branches must also be generated in ETAR. Instead of choosing the branch attribute from a list of explicit attributes, complex matching conditions are used on traces of the examples. When portions of the trace remain unmatched, a branch is found. It must be explained from the attributes present in the object hierarchy and related to the example through the FOA.

SPARC/E — instantiation of abstract skeletal rules

The SPARC/E [Dietterich & Michalski 1986] system addresses issues in sequencing and data transformation. These are important in task learning. ETAR also accepts numeric sequences and transforms them to fit a symbolic model before major generalization can occur. SPARC/E is a Sequential PAttern ReCognition system developed specially for finding rules in the card game Eleusis. An example is a partial sequence of objects (mainline) along with instances of incorrect entries to the sequence (sideline), shown in figure 3.4. This sequence must be matched to one of SPARC/E's three types of rule schemata. A decomposition schema allows rules as a set of implications which depend on previous elements of the sequence. Rules with repeating patterns are determined by a periodic model. Finally, a disjunctive normal model generates rules as a collection of conjuncts. As illustrated in the figure, four steps determine a rule:

1. Choose one of the rule models and fix its parameters (such as the length of a period or the number of previous elements on which the rule may depend).
2. Transpose the example sequence into a form suitable for analysis by the methods associated with the chosen schema.

Mainline: **JC 4D QH 3S QD 9H QC 7H QD ...**
Sideline: **KC 5S 4S 10D**
7S

1. Choose model: periodic with period of 2
2. Transpose sequence:
   [JC QH QD QC QD]
   [4D 3S 9H 7H]
3. Generalize: period(faced/nonfaced)
4. Verify: all sideline elements are excluded

Figure 3.4: SPARC/E example (adapted from [Dietterich & Michalski 1986])

3. Generalize the transformed example, based on the positive evidence, so that it fits into the schema.

4. Verify the rule to ensure that it covers all positive examples and no negative ones.

3.2.2 Function Induction

Some Machine Learning systems construct functions from examples. Whereas the systems of the previous section identify sets, the systems in this section find a relationship between sets. This section introduces function terminology and function induction, and provides two exemplar systems.

A function, f, is a mapping from a set I to a set O which assigns to each $i \in I$ a unique value $o \in O$. I is the domain and O is the range of the function. Generally, I and O are cross product sets, where $I = I_1 \times I_2 \times I_3 \times ... \times I_m$ and $O = O_1 \times O_2 \times O_3 \times ... \times O_n$. This is written as

$$f : I_1 \times I_2 \times I_3 \times ... \times I_m \longrightarrow O_1 \times O_2 \times O_3 \times ... \times O_n$$

A function, f, can be considered as a coordinate tuple of functions, that is, $f = (f_1, f_2, ...f_n)$, where each $f_i : I_1 \times I_2 \times I_3 \times ... \times I_m \longrightarrow O_i$.

Given any set of input–output tuples, function induction determines a function which relates the inputs to the outputs. Finding this function, f, reduces to the problem of finding each coordinate function, f_i. This problem, as it stands, is unsolvable since the space of possible functions is infinite as is the world variable space. Thus, numerical analysis and approximation theorems[4] alone do not suffice in real world problems. Function induction methods search for the desired function and their goal is to determine constraints which make this search feasible. Two function inducers are now described, which like ETAR, must first limit the relevant variables.

ABACUS — proportionality relations on subsets

ABACUS [Falkenhain & Michalski 1986] looks for monotonic relations between the possible variables. The system generates a split function, with breakpoints determined whenever monotonicity changes. Generating the applicable function on each region proceeds by applying compositions of four functions ($\cdot$, /, +, −) to the variables in an attempt to obtain a constant result. The monotonic relation limits the actual functions and variables investigated — if x mostly increases as y increases try x/y or $x - y$; if x mostly decreases as y increases try xy or $x + y$. Variables different from x and y must be held constant for this check. Thus a huge number of examples are required. Moreover, these examples must be well chosen so that each variable remains constant a sufficient amount. With ETAR this is impossible since the user is not aware of the underlying induction and cannot be expected to present appropriate kinds of examples. ABACUS also constrains its search by checking that the physical units of variables are compatible for each function application and that algebraic constraints of the four operators are considered. Algebraic duplicates are eliminated by using canonical forms and avoiding actions which result in mathematical cancellation. ETAR, too, must avoid algebraic duplicates.[5] Once ABACUS has discovered the function, its applicability conditions (often non–numeric) are discovered by the

[4] *E.g.* The Stone–Weierstrass Theorem [Bartle 1976]
[5] This is one of the next steps to be implemented.

Law of Falling Bodies: $S = Vt + \frac{gt^2}{2}$

primitives:
units: meters m, seconds s
operators: multiplication, exponentiation
variables:
independent: velocity V (m/s), time t (s)
dependent: distance S (m)
examples:

V	t	S given	S calculated
6	2	32	32
10	12	840	320.4
5	2	30	26.7

Figure 3.5: Example of COPER (adapted from [Kokar 1986])

A^q approach discussed earlier.

COPER — dimensional analysis

COPER [Kokar 1986] is a discovery system which uses polynomial approximation. Relevant variables are established by forcing the dimensions or units of the independent variables to match those of the dependent variable. The user provides the set of primitive units and primitive operators or rules for combining the units, as shown in figure 3.5. Function generation follows four steps:

1. Select a basis, of size equal to the number of primitive units, from the independent variables.

   In the example there are two units, so choose a two element basis.

   There is only one choice: $\{V, t\}$.

2. Express all other variables as a composition of a constant and primitive operators between basis elements. The constant is determined from an initial, observed set of values for all the variables.

   There is only one other variable, S, in the example.

   $$S = cVt.$$

Substituting the initial observation gives

$$S = 2.67Vt.$$

3. Vary the values of the basis elements while keeping the non–basis, independent variables fixed. Use the results from the previous step to calculate the dependent variable.

   Since there are no non–basis, independent variables in the example, try any combinations of V and t.

4. If the calculated value of the dependent variable is the same as the observed (or given) value, then the set of independent variables is complete so polynomial approximation is used to generate the desired function. Otherwise, some independent variables are missing and must be determined. COPER generates possibilities for these by combining primitive units.

   In the example there is an independent variable missing, since the observed S values do not match the predicted ones. Once g is included COPER generates the function.

COPER's approach is not feasible for ETAR's function induction for two reasons. First, COPER has no way of disregarding irrelevant variables. This is a major consideration for ETAR since there will be many variables with the same dimensional units, *e.g.* height, width, length. Second, COPER needs to be given actual values when it performs the relevancy test. ETAR's user cannot provide these. Hence, ETAR contains the FOA mechanism to partially circumscribe the variables and then relies on a straight forward generate and test procedure.

3.2.3 Automatic Program Construction

Automatic program construction (APC) takes a variety of forms [Biermann *et al* 1984], but this section focuses on the synthesis of program code from examples of the desired behavior. Originally, the programs were intended to be in computer

primitives:
car, cdr, cons, nil, identity
functional form:

$$f_i(x) = (\text{cond } (p_{i1}\ \ f_{i1}) \ (p_{i2}\ \ f_{i2}) \ \vdots \ (p_{in}\ \ f_{in}))$$

where p_{ij} is the LISP atom predicate operating on a composition of car's and cdr's
f_{ij} is a function composed of the primitives.

an example: Given a list, return a list of the first element
(A B C D) → (A)

Figure 3.6: Regular LISP programs

languages, but success has only been with small subsets of functional languages (*e.g.* LISP and PROLOG [Biermann *et al* 1984]) or scant pseudo–languages [Amarel 1986]. Under these restrictions, it is difficult to distinguish APC from function induction, aside from the authors' preference. Looking at this preference, the main distinction is that the search in APC is limited by modelling the desired program with an external flow control structure. Such structure is not attained in function induction. Biermann's work, the most successful APC, illustrates this.

Biermann's automatic LISP program construction

Biermann divides programs into classes and finds class–specific methods for APC. This avoids controlling the search with general heuristics which often result in unreliable programs and allows constructing programs through a minimum number of examples. Biermann [Biermann 1984] provides techniques for two classes of LISP programs: regular and scanning. The regular programs are summarized here.

Regular LISP programs are composed of five primitives and a conditional form as illustrated in figure 3.6. From examples (often only one), two stages are necessary to generate the regular LISP expressions:

1. composition stage — expresses the example as a sequence of the five primitive functions. First the example is written as a composition of the primitive functions using composition rules associated with the regular LISP model.

   The example (from [Biermann 1984]) in figure 3.6 can be decomposed as follows:

   $f_1(x) = \text{cons}(f_2(x), f_4(x))$
   $f_2(x) = f_3(\text{car}(x))$
   $f_3(x) = x$
   $f_4(x) = nil$

   Further rules break the composition into a unique trace, without redundant, useless operations, and so that car and cdr operations have precedence over other primitive operations.

   In the example this is:

   $f_1(x) = f_2(\text{car}(x))$
   $f_2(x) = \text{cons}(f_3(x), f_4(x))$
   $f_3(x) = x$
   $f_4(x) = nil$

2. generalization stage — merges the individual functions from the trace into as few functions as possible. A merger succeeds when conditional predicates can be generated between functions, resulting in a conditional form function. This is attempted between all possible combinations of functions in the trace, starting with the trial to merge all.

   In the example, it turns out that only two functions can be merged: f_1 and f_2. This gives

   $f_1(x) = f_1(\text{car}(x))$
   $f_1(x) = \text{cons}(f_3(x), f_4(x))$

where the non–determinism must be explained by conditional predicates, to yield the final program:

$$f_1(x) = \text{cond}(\text{atom}(x) \rightarrow \text{cons}(f_3(x), f_4(x))$$
$$t \rightarrow f_1(\text{car}(x)))$$
$$f_3(x) = x$$
$$f_4(x) = nil.$$

ETAR is also a form of automatic program construction, for robots. It also uses a model, now of robot tasks. The model is generated by the PMFs, the FOA, and the conditional constructs. But ETAR stands apart from APC for two reasons:

1. ETAR controls an environment which is external to the robot and the computer
2. sequencing is essential and explicit in ETAR but never considered in current APC.

Figure 3.7 summarizes and compares the learners discussed in this section. It states the background knowledge required, form of the learning examples, result of learning, and underlying search control mechanisms.[6] Note that none of these systems work with external environments.

3.3 Human Task Learning Theories

Since this thesis is concerned with task learning, it is worthwhile to consider the theories in human task learning. Several of these theories propose that humans tasks are *programs* and so learning a task is equivalent to learning a program. This section points out how these programs differ from what ETAR learns. Human tasks are generally termed "motor skills" or "movement forms" and task learning is the acquisition of motor skills so is often called motor learning. It is assumed that movement forms arise from one of two cases [Schmidt 1982]:

[6] See also [Witten & MacDonald 1988] and [Dietterich 1983] for summaries of other concept learning systems and comparisons with different criteria.

system	background knowledge	form of examples	what is learned	method of control
Version Space	partially ordered language, often containing attribute vectors	members of the concept hierarchy, with an indication of whether the member is a positive or negative example	given an object, can classify it as to whether it is an example of the concept learned	hierarchy traversal where negative examples cause convergence down and positive examples cause convergence up
A^q	attribute sets	set of positive and negative examples of the concept	discrimination conditions to distinguish positive examples from negative ones	beam search in comparison of positive and negative events
ID3	none	complete set of examples represented as attribute vectors with an indication of the category the example belongs to	given another object, can classify it	information theory to control the generation of branches during the construction of a decision tree
SPARC/E	list of relevant attributes	sequence of objects, with an indication of whether the object is in the correct position or not	rules for characterizing elements in this particular sequence	schema instantiation, hence limited by the type of schemata initially present
ABACUS	structure of data types	input–output tuples	split function of possibly several variables	proportionality relations on subsets canonical math forms A^q techniques for conditionals
COPER	primitive units (e.g. meter, second, ...) operations for combining these	independent variables and dependent variable with units and values	function relating (a subset of) the independent variables to the dependent variable	dimensional analysis polynomial approximation
Biermann's LISP APC	LISP primitives, production rules	generally one example, represented as an input–output data association	LISP program which generates correct output for the example(s)	composition/ trace model production rules

Figure 3.7: Comparison of systems discussed in this section

1. genetically defined, *e.g.*, limb control, reflex actions.

2. learned, *e.g.*, throwing a ball, driving a car.

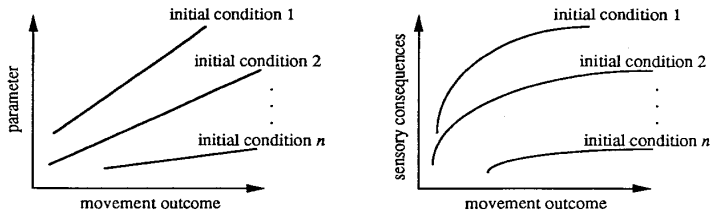
Some tasks have always been known; others must be acquired.

For many years, motor skill acquisition has focused on Thorndike's theory of reinforcement learning[7] [Schmidt 1982]. The theory insists that those movements with favorable response are remembered while others are not. This was the accepted view for a long time, and often appears as an underlying element in other learning theories. Adams' [Adams 1976] closed-loop theory and Schmidt's schema theory are currently the two prominent human task acquisition theories. These are summarized in this section. Neither fully explains human task learning, but they do illustrate the issues, difficulties, and lines of thinking that are present in such theory development.

Adams' Closed-Loop Theory

Adams proposes that performing tasks is a closed-loop process which uses feedback, error detection, and error correction [Adams 1976]. Error detection and correction is based on the previously learned task, called a *reference of correctness* or *perceptual trace*. Learning a task is learning a perceptual trace. This is attained by starting a motion, and then using knowledge of the results[8] to guide the limbs to the correct positions. Stimuli of the limb locations "leave a trace" in the brain. Each trial generates a new trace and the collection of traces makes up the perceptual trace whose strength increases with each trial. Thus, errors have a drastic effect, so guidance is essential in the learning stage.

[7] Also known as the theory of reward and punishment or the law of effect.
[8] A verbalizable form of extrinsic feedback, often abbreviated to KR.



(a) Recall Schema (b) Recognition Schema

Figure 3.8: Task learning in schema theory (adapted from [Schmidt 1982])

Schmidt's Schema Theory

Shifts away from reinforcement interpretations use the information present before, during, and after an action [Newell 1981]. Schmidt's schema theory [Schmidt 1982, Schmidt 1976] falls in this category. In this theory, a task is two schemata, whose domination is task dependent:

1. recall — generates the impulses to the muscles for movement or movement correction.

2. recognition — evaluates the feedback produced by the response.

Underlying these schemata is the idea that all actions are *generalized motor programs*, containing variables such as duration, phasing, and relative force, which are stored in the central nervous system. A task is executing the appropriate program with its variables properly instantiated. Schmidt makes no claims to indicate how the motor program is obtained. Some possibilities may be guidance, observation, or analogy. In schema theory, learning a task is equal to acquiring a recall and recognition schema to control the generalized motor programs.

The recall schema is acquired through trials — the initial conditions are known, values for the parameters are chosen, and the motor program is executed to determine the movement outcome. After some time, a relationship (the recall schema) is determined between the parameter values, the initial conditions, and the movement

outcome, as shown in figure 3.8(a). Incorrect outcomes do not hinder the acquisition process. Rather, provide a data point to be used in generating the schema. Once the recall schema is learned, a task is performed by mapping the initial conditions and the desired outcome into the schema to determine the parameter values for the generalized motor program, which is then executed. Learning the recognition schema proceeds in a similar manner [see figure 3.8(b)].

Iba [Iba & Langley 1987] is simulating a similar theory on a computer. ETAR is not an implementation of human skill acquisition theories; rather it is an algorithm to make acquisition possible in robots. Nevertheless, there are interesting similarities. The most notable is that the initial conditions are mapped into the movements to determine the final outcome, which is a program. In schema theory there is no indication of which initial conditions effect the outcome. In this thesis, only conditions in proximity to the robot gripper are relevant.

3.4 Learning by Example in Robot Domains

Machine Learning researchers tend to avoid task learning. Whenever coordination with an external environment is necessary, problems are abundant. Issues such as path planning, error recovery, accuracy, sequencing, and handling large amounts of information are prominent and unanswered. Three systems, ARMS, DIFFY-S, and NODDY, begin to deal with some of these problems. This section looks at the type of world each system operates in, the type of learning that occurs, the form of examples the user must present, any background knowledge required, the control structures handled, and how variables are incorporated. Focussing on these issues, figure 3.9 provides a comparative summary of the three systems together with ETAR. To my knowledge, ARMS is the only one of these that has been implemented on a real robot arm, and hence is a true task acquisition system. The other two systems operate in simulated workspaces. Nevertheless, they provide a methodology which must be transferred to real robots.

	ARMS	DIFFY-S	NODDY	ETAR
robot world	implemented on a MicroBot Teachmover in an assembly domain	system for learning operators which could then be applied as in STRIPS	simulated point robot in a 2-d world containing objects	implemented on an Excalibur robot in a blocks world
type of learning	explanation-based	inductive	inductive	inductive
form of examples	expert leads through one assembly	initial and final world state and all parameters	teacher commands point robot	user leads robot through examples of task
background knowledge	extensive frame system with procedures for object manipulation	primitive function schemata	primitive functions and motion hierarchy	concept hierarchy of frames and primitive functions
control structures	sequencing (various conditions are handled by domain procedures)	conditionals	sequencing, loops, and conditionals	sequencing, loops, and conditionals
variables	no variables introduced	variable choices must be explicitly stated	variables discovered by finding relations between locations	variables discovered from the focus of attention in a task

Figure 3.9: Comparative summary of four robot learning systems

3.4.1 DIFFY-S — simulated robot operator learning

DIFFY-S [Kadie 1988] is an operator learning system which learns from examples of the operator's desired or observed effects. Though DIFFY-S learns operator schemata which may be used by robot systems, such as STRIPS [Fikes & Nilsson 1971, Fikes 1972], it never involves the robot during the learning process. Examples are presented as a nested tuple of nominal values, numbers, and data types, and are snapshots of the world before and after operator application. Each example must explicate the object to which the operator is applied and the set of attributes which may influence the outcome.

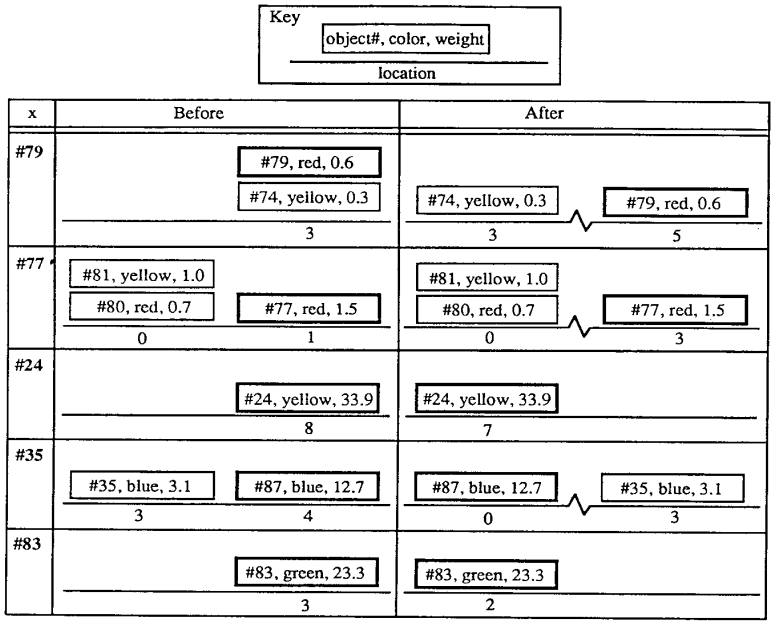
The operator learned is a split–function, expressed as a nested "if-then-else" structure. Conditions are generated from the differences in attribute values between examples, and the corresponding explanation is composed of primitive, background functions. The precise manner in which this is done is unclear in the publication and the implementation is not yet finished. Briefly, the learning algorithm contains three parts:

1. explainer — generates functions to explain the operator for a subset of the examples. This is accomplished by generating all possible combinations of the primitive functions in a depth first manner using type constraints. ETAR uses a similar search, only it is breadth first and constrained by the FOA.

2. flattener — converts the nested tuple world representation to a simple tuple containing the relevant attributes.

3. concept learner — determines the conditions for distinguishing each subset for which the explainer has found a function.

Figure 3.10 exemplifies DIFFY-S. Part (a) is a table illustrating a domain and five examples presented to system, while (b) is a paraphrase of the LISP operator constructed from them.



(a) Examples of **op**(x) presented

op(x) =
```
if weight(x) ≤ 5.2
then move(x) to(location + 2)
else if weight(x) ≥ 23.3
     then move(x) to(location − 1)
     else move(x) to(0)
```

(b) Operator learned from the examples

Figure 3.10: DIFFY-S learning an operator, **op**

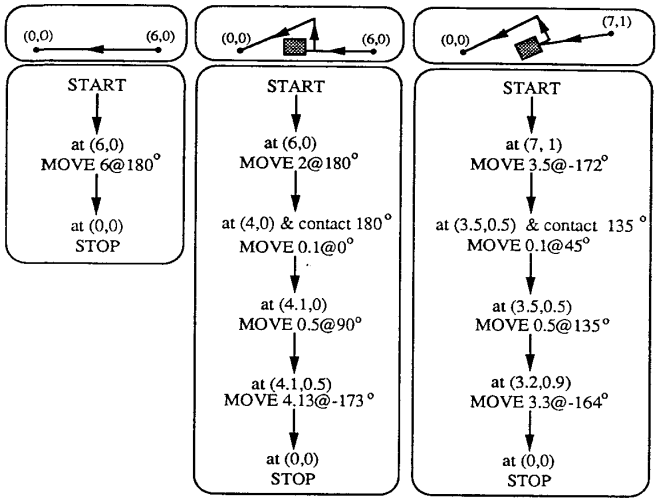


Figure 3.11: Examples of a task — moving around an obstacle — presented to NODDY (from [Andreae 1984])

DIFFY-S is a beginning in automatic robot operator construction from examples of the task. However, expansion in several directions is necessary for real world activities. Often tasks comprise steps which are repeated, hence require a loop construct. The focus of attention may change throughout a task or may contain more than one object. Instead of requiring the user to elucidate the object of attention and its important attributes, any system operating in dynamic environments must determine these on its own. Specification of examples as attribute–value lists is difficult and must change so that the robot is used. ETAR makes these improvements.

3.4.2 NODDY — learning procedures for a point robot

From examples, NODDY [Andreae 1984, Pauli & MacDonald 1988] learns procedures in the simulated world of a two-dimensional point robot. A teacher inputs examples

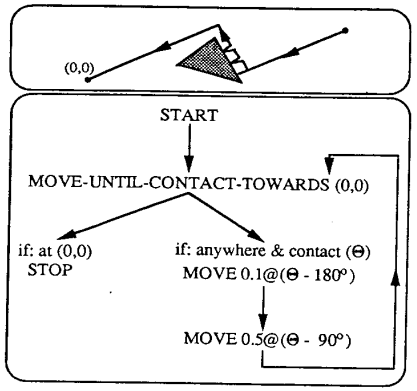


Figure 3.12: Procedure learned from the examples in figure 3.11

as traces containing nodes connected with arcs for flow control. Each node is an event and consists of an action and an optional condition limiting when the action may occur. Both the condition and the action may contain parameters. Three examples of a typical NODDY task are shown in figure 3.11, while figure 3.12 contains the procedure which NODDY constructs from them. Procedures are constructed incrementally, so generalization occurs after each example. Six stages shown in figure 3.13 make up this process:

1. include — checks to see whether the example trace contributes new information to the procedure. If not, a new example is awaited.

2. skeleton — matches key events based on the action (but not the parameters to the action) description in the nodes of the trace. These are events whose action occurs only once in the trace and once in the current procedure. ETAR also has key event matching. But since unique events are rare, key events are determined from the FOAs and PMFs.

3. propagation — pairs nodes between the trace and the procedure by matching events up and down from the key ones, provided there is justification. This

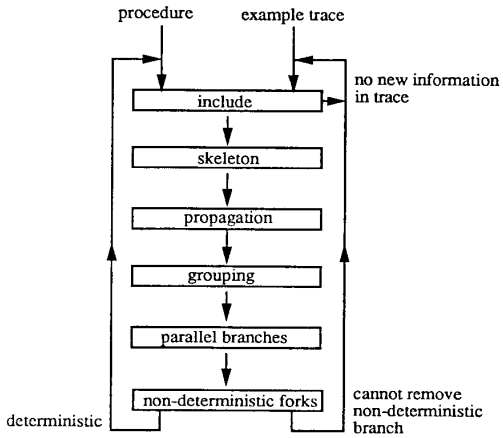


Figure 3.13: NODDY organization

justification is a set of domain dependent heuristics which force the pairing process to depend on the actions, the conditions, and their arguments. An example justification may be that corresponding actions terminate with the robot in the same position. ETAR also uses strong justification conditions — a key event must match to another and have a similar focus of attention. But ETAR matches events only by a sequential pass through both the procedure and the new example.

4. grouping — merges connected linked events into one event using the domain dependent action hierarchy. Loops automatically appear when two trace events are matched to a single event in the procedure. No action hierarchy is present in ETAR. Furthermore, ETAR generates loops within a single example.

5. parallel branches — are combined by inducing functions to relate the parameters in the conditions and actions. A parallel branch is recognized by having coincident starting and ending nodes, the same number of events, and matching conditions and actions for corresponding events. The function induction

stage uses a set of predefined semi–invertible[9] binary and unary operators. A function between the input, determined from the condition parameters or previous events, and the output, or parameters to the action, is required. Search proceeds by applying compositions of the operators to the input and inverses of the operators to the output in an attempt to eliminate the difference. New constants are generated in special cases as the last step in this induction. If a binary operator, when applied to the input and output, always yields a constant value, then the semi–inverse (containing the constant) is the unary operator which completes the desired function. ETAR's function induction algorithm proceeds in only one direction. Thus, the ability to induce a constant is lost. However, required constants are assumed present in the knowledge base.

6. nondeterministic forks — ensures that the constructed procedure is deterministic. If it is not, merging, prioritizing, and quitting are some remedies.

The full capabilities of NODDY are unknown; limitations are more obvious. Presentation order of the examples is important, placing a heavy onus on the teacher. Setting up the input traces and domain knowledge is difficult, especially knowing when the contact angle must be given and when the path angle is more important. Its feasibility would be greatly increased with an interface which constructs the traces from the teacher leading the robot, as this thesis proposes. Long tasks are hard to process since the skeleton matching phase looks for key (unique) events which become fewer and fewer as the task traces increase in size.

All problems aside, NODDY is an impressive system which learns procedures with sequencing and loops. Additionally, these procedures may contain variables. In NODDY, these variables are discovered from previous events. This thesis proposes an alternative for finding variables, namely, that the variables arise from characteristics

[9]For unary operators, semi–invertible is the same as invertible. A binary operator, $f(x,y) = z$, is semi-invertible if there exist two operators g and h such that $g(y) = g(y,z) = x$ and $h(x) = h(x,z) = y$, for all fixed z. In the description here, both g and h will be called semi–inverses, though Andreae [Andreae 1984] refers to them as inverses.

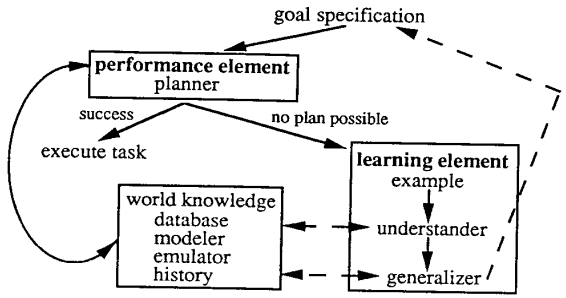


Figure 3.14: ARMS organization

of the objects being manipulated. Furthermore, in addition to using key events to generate loops, these manipulated objects form the basis for inferring loops and conditionals, and for restricting the function induction.

3.4.3 ARMS — Machine Learning with a real robot

ARMS [Segre 1988] applies explanation–based learning [DeJong & Mooney 1986, Mitchell *et al* 1986b] to the robot retraining problem. Unlike most Machine Learning systems, ARMS has actually been implemented on a real, five–degree of freedom robot (a MicroBot Teachmover) with moderate success [Gustafson 1986].

The structure of the ARMS system is shown in fig 3.14. As in all explanation–based systems, ARMS consists of two components which depend heavily on a well–organized knowledge base:

1. performance element — Here resides the planner, or heuristic problem solver, which uses the domain knowledge to determine a sequence of executable operators to transpose the initial world state into the desired goal state. When this is impossible, it is assumed that the domain knowledge contains the required functional information, not yet in an operational form. For example, "make a revolute joint from x and y" implies that the objects, x and y, must be placed

in simple rotation, about a single axis, with respect to each other. This is only a functional description since there is no awareness of all objects or the operation sequence required to carry out this task. In this case, control diverts to the next component.

2. learning element — From an initial state, a goal specification, and an observed sequence of operators, an operational description of the task is constructed. This is accomplished in two steps. First, the understander builds a causal model of the low level operators' constraint criteria. This model is passed to the generalizer which verifies that the observed episode satisfies the goal specification by constructing an explanation beginning from the initial state. Provided the preset learning criteria are met, the generalizer then constructs the new operational description for the task. This employs generalization and specialization.[10] Generalization abstracts the explanation using techniques such as replacing instances by variables. Other times it is necessary to constrain the original example, by specialization, to aim for less effort and to gain efficiency.

Figure 3.15 exemplifies a typical ARMS domain and task. The background knowledge contains information about joint–relations between assembly objects. The initial state must be given by stating the physical pieces in the workspace along with their positions relative to the workspace frame. Additionally, the functional goal must be specified. Determining the operational task description, *i.e.* the mating conditions and robot motions, is the duty of ARMS. Because examples portraying the desired functional properties and sequences of operations for assembling these workspace objects are both unknown, stage one (the planner) fails. Hence, the learning phase is invoked. The expert constructs the revolute joint by a sequence of motion primitives, which the understander converts to a causal model. Finally, the example is generalized so that a revolute joint may be efficiently constructed in a

[10] General explanation–based learning may also use refinement techniques to make modifications to the explanation or previously learned concept. ARMS does not.

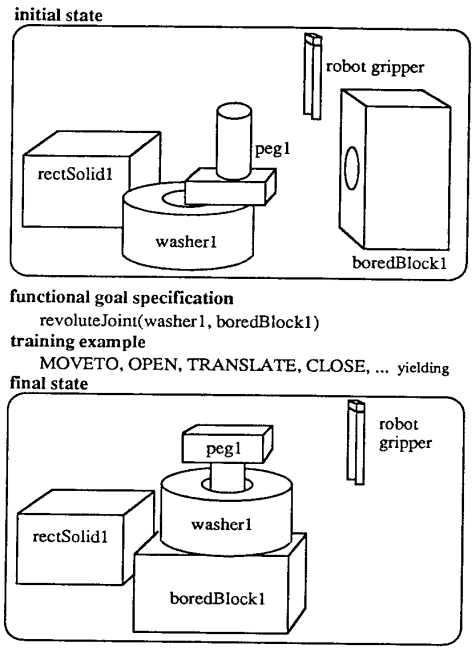


Figure 3.15: Example presented to ARMS

variety of circumstances, which do not depend on the presence or use of extraneous objects, such as rectSolid1.

The background information and procedures are the bread and butter of ARMS. The primary feature is the database, which is expressed as a schema system and contains descriptions of the physical objects, relations in the world, and operations which can be applied. As the system operates, it relies on a graphic model, emulator, and history mechanism to maintain the world description. It assumes that motion is instantaneous, does no collision detection and little collision avoidance, moves only one object at a time with perfect contact points between objects, and always places

one object perfectly on top of another.

ARMS is an important contribution to machine learning and robotics. It achieves its goal — enabling a robot to adapt for known tasks within new environmental configurations or to be retrained by a single example. ARMS illustrates real world constraint–based generalization and modelling. The biggest problem is that a large amount of domain knowledge, both passive and active, is required. ARMS is not capable of acquiring new knowledge without the background functional description. Many nonprimitive operations such as *stack* and *unstack* must be entered by a robotics expert. Hence, whenever new or modified objects appear in the environment, these operations must be recoded, again by the robotics expert. ETAR aids the general user in these situations since it is precisely these sorts of tasks which the user should be able to teach a robot. ARMS also needs improvements in the user interface, namely vision systems and teaching pendants, to make the input of the initial state and task example feasible.

3.5 Recap

The chapter began with two indistinct dichotomies of knowledge shown in figure 3.16. Then it described numerous machine learning systems and showed their relation to ETAR. Two human task learning theories were also presented. As was obvious in the discussion and as is depicted in the figure, Machine Learning researchers tend to avoid open knowledge. The top half, the open knowledge portion of the figure, is virtually empty — Adams and Schmidt's theories are human learning theories, so that only leaves ARMS and the thesis system. Now details of the ETAR methodology begin.

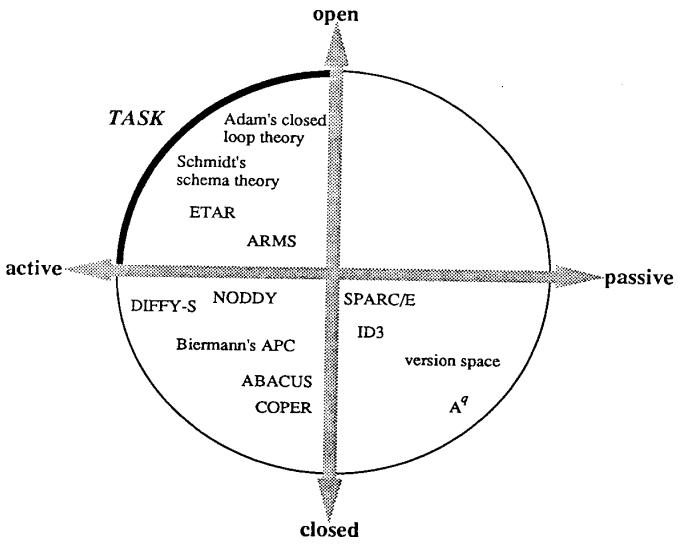


Figure 3.16: Knowledge types for systems and theories discussed in this chapter

Chapter 4

Robots Acquiring Tasks from Examples

Recall that ETAR learns tasks from numeric examples obtained when a user leads the robot. Examples are converted into a symbolic form consisting of *primitive motion functions* (PMFs) and generalized into a task description. The *focus of attention* (FOA) mechanism forces ETAR to concentrate on "important" parts of the task and hence to form loops, match examples, and limit the scope of task variables. Function induction fully merges all examples.

This chapter explicates the robot task acquisition system [Heise & MacDonald 1989b, Heise & MacDonald 1989c]. Figure 4.1 depicts the system, highlighting the main chapter sections. First, the learning examples are described. Background knowledge of the workspace model is presented next. Section three discusses how the input examples are converted from numeric feedback into a partially symbolic form. Generalization of the symbolic forms comprises sections four and five. In section four, generalization at the motion name level is detailed while section five describes generalization of the arguments to these motions. A summary appears in the final section. Where applicable, the examples from chapter two serve to illustrate the acquisition process.

4.1 Input to the Learner

From the user's perspective, an input example is "the result" of physically leading the robot. Instead of computer programming and robotics knowledge, the only expertise required is the ability to lead the robot in the task.[1] The user *sees* the task as it is

[1] Or, in the Excalibur, use a model of the robot for the task.

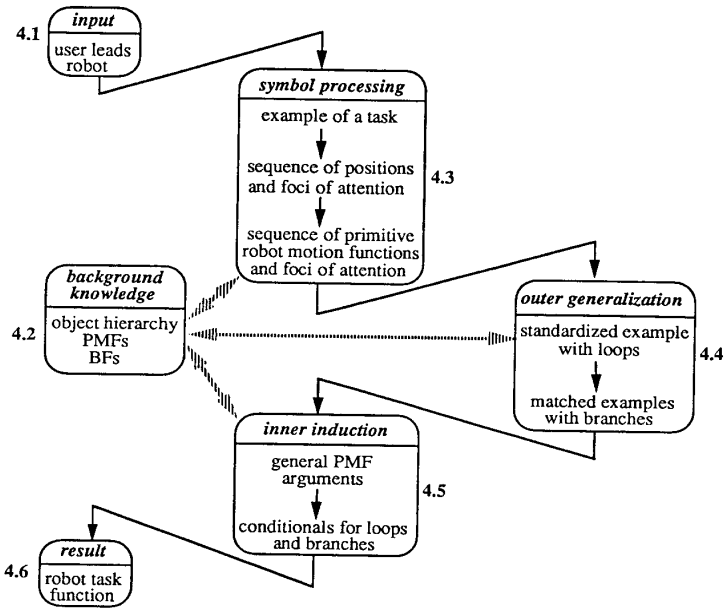


Figure 4.1: The ETAR task acquisition system

accomplished and thus *sees* the example.

From ETAR's perspective, an example is numeric feedback sampled from the robot to indicate the path followed. Available feedback, especially sensory information, varies in each robot system. The Excalibur samples[2] at 10 *Hz* and each point is a hexadecimal string which decomposes to the six joint angles and gripper position.

4.2 Background Knowledge

ETAR contains three kinds of background knowledge:

[2] This rate can be changed at the robot controller level, but the thesis implementation uses 10 *Hz*.

1. PMFs from which the robot's programs are composed.
2. Basis functions for the late stages of the induction.
3. Workspace.

Clarification of PMFs and basis functions is postponed to their occurrence in the learning algorithm. The major background knowledge for ETAR is in the workspace representation. Individual objects must be expressed so that their properties and common characteristics are easily obtained. Furthermore, as the robot moves through its workspace, the nearby objects must be evident. This section concentrates on these issues.

4.2.1 Object Hierarchy

Any language for representing a robot workspace in the ETAR system must be expressive enough to:

1. contain variables, constants, and procedures which, together, describe workspace objects. It is the variables and constants associated with an object that provide the domain for the final stage of task construction. Procedures indicate uses for the object, and are often tasks which can be learned by ETAR.
2. highlight the similarities and differences between workspace objects. Similarities and differences impose major constraints during the search for functional relationships in task construction.

Frame systems [Minsky 1977] fulfill these requirements. A frame is a data structure which is referenced by the object's name and contains aspects (slots) and values (fillers) about the object. These values may be constant, variable, or procedural, hence satisfying the first language requirement. Each frame contains special slots for subtypes and supertypes which impose an order relationship or hierarchical inheritance mechanism, fulfilling the second language requirement.

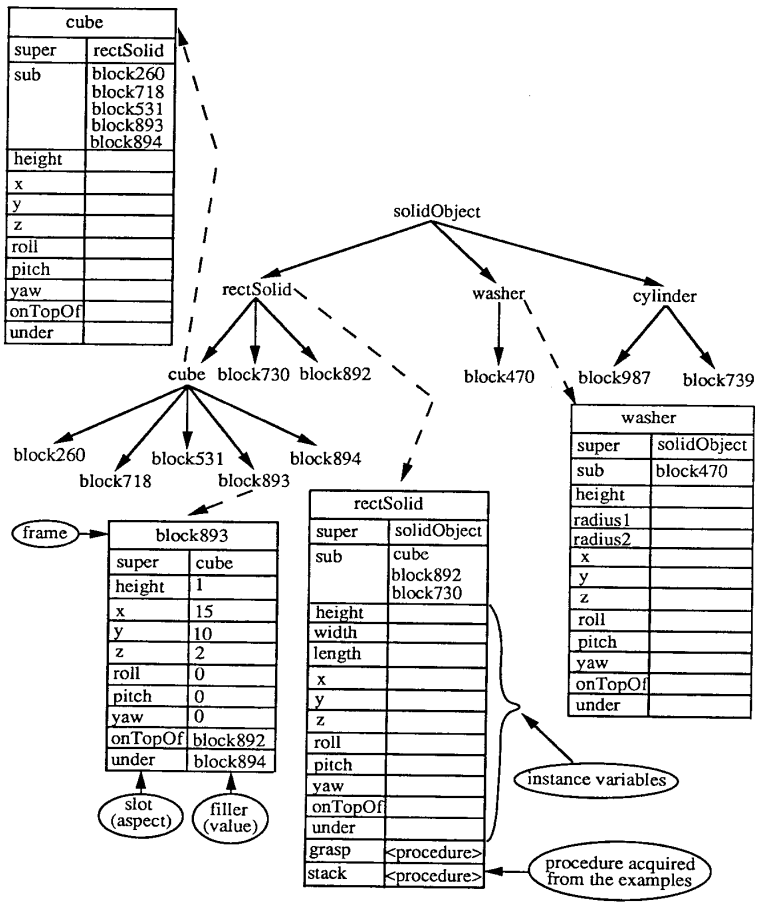


Figure 4.2: Workspace objects represented as frame system

A frame system represents ETAR's workspace knowledge. A partial example is in figure 4.2: the tabular portions are the data stored in ETAR; the visible hierarchy just elucidates inheritance between objects. Each workspace object is a frame, *e.g.* block893, which is an instance of (and subtype of) a more general object such as a cube, which is in turn a subtype of an even more general object. Thus, the workspace knowledge is referred to as an *object hierarchy*. Each object in the hierarchy consists of many slots and fillers. The fillers of general objects may be empty, *e.g.* the value for height in rectSolid is unknown, but must be present for distinguishing instances. ETAR's implementation uses SCOOPS,[3] which provides the underlying mechanisms, including inheritance, to maintain frame systems.

Entering the background knowledge, especially ensuring that all necessary aspects are present and correct, is difficult for robot users. Any objects with which ETAR operates must be simplified into a frame representation, thus effectively limiting the domain to which it can be applied. Future work must investigate solutions to these problems. Until then, the information for these frames is generally present in computer-aided design [Knox 1984, Medland & Burnett 1986, Teicholz 1985], so it is not unreasonable to expect its availability here.

4.2.2 Workspace as Grid

Knowing which objects are being manipulated is crucial to ETAR's success. The onus for pointing out these objects could be on the user, but this is not viable. In the future, a vision system must distinguish these objects. For now, a workspace model suffices. This model is in addition to the object hierarchy, and though it contains no new information it indexes the hierarchy by position. This is essential when the object hierarchy is incomplete or extremely large, since then an exhaustive search through the hierarchy is unsuitable.

[3]SCheme Object Oriented Programming System [TIScheme 1987].

There is a cube [block65] of height 1″ stacked on top of a cube [block72] of height 2.5″ at (3.8, 2.0).

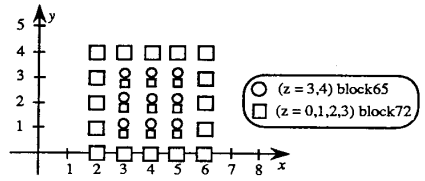


Figure 4.3: Example of workspace grid

To aid the FOA mechanism, this model is implemented as a two-dimensional integer grid representing the x-y surface of the workspace. An object's reference name along with its z-location is placed in the grid at all locations the object occupies. A real valued location is approximated by the integers spanning it.[4] Rounding the values in this manner does not affect task construction, since the actual values are always retained in the object's frame. Figure 4.3 depicts a small part of a workspace.

In addition to the grid, the workspace also contains a set of procedures for manipulating it. This includes procedures for initialization, display, movement, addition and deletion, and determination of objects in the FOA. Due to the importance of the latter procedure, it is discussed as a major part of the next section.

4.3 Symbol Processing

The first step in the learning algorithm converts the robot position feedback into symbolic form augmented with numeric arguments. This section details this conversion — changing to world coordinates, indexing the workspace to determine FOAs, circumscribing the teaching sequence using the FOAs, and finally extracting PMFs.

[4]*E.g.* −1.3 maps to [−2, −1] and 1.3 maps to [1, 2].

4.3.1 Robot Position Feedback to World Coordinates

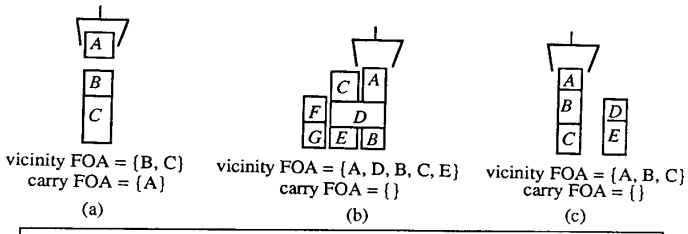
Transforming the position feedback into world coordinates is a complex geometric and algebraic process. The Excalibur returns teaching sequences as hexadecimal joint angles which are converted into decimal angles and then passed to a kinematics routine to result in the corresponding world position: the location (x, y, z), an orientation (*roll*, *pitch*, *yaw*), and the gripper position (*open/closed*).[5] Complete discussion of kinematics is deferred to chapter 5. Once the robot path is traced as world positions, workspace indexing extracts the FOA.

4.3.2 Focus of Attention

ETAR assumes that the objects closest to the robot gripper are the important ones, *i.e.* the FOA. The FOA mechanism discloses and suppresses objects as they come in and out of play during the task. Two FOA categories exist and are illustrated in figure 4.4(a, b, c) for a two–dimensional environment:

1. *carry* — objects which the robot is manipulating during a motion. With the Excalibur, this is the object which the robot is grasping, since ETAR assumes that the robot will not use objects as tools to move others, nor that more than one object will ever be grasped. Other robots, *e.g.* HERMIES–IIB [Weisbin *et al* 1989], may grasp one object in each arm and pile other ones onto its base, thus having multiple objects in the carry FOA. Robots capable of ballistic motions, *e.g.* [Andersson 1987], may manipulate objects with primitives different from grasp, such as "hit".
2. *vicinity* — objects within a certain distance, called the *vicinity distance*, of the robot's gripper but which are not in the carry FOA. The thesis implementation sets the vicinity distance to three inches since it is working in a static manip-

[5]The Excalibur really has three values for the gripper: 1 indicates open, −1 closed, and 0 no change. The 0 is ignored.



LET (x, y, z) be the robot hand position
LET (x_o, y_o, z_o) be an object position
LET hdist(o) be max($| x - x_o |, | y - y_o |$)
LET vdist(o) be $|z - z_o|$
LET vicinity distance be 3 inches
LET max hdist be 0 inches

WHILE (max hdist $\leq$ 1) or (vicinity FOA is empty and
max hdist $\leq$ vicinity distance)
FOR ALL objects
if vdist(o) $\leq$ vicinity distance and hdist(o) $\leq$ max hdist
then add[a] object o to the vicinity FOA
INCREASE max hdist by one inch

[a]objects are ordered first on hdist and then on vdist

(d) vicinity FOA mechanism

Figure 4.4: The focus of attention

ulative environment. If the environment were dynamic, *e.g.*, in a ping–pong game, the range would need to be larger and other characteristics, such as the speed of objects or rules of the activity, could be used to constrain the FOA.

Objects in the vicinity FOA are identified by searching a neighborhood of the robot gripper.[6] An "importance" ordering is imposed to signify the physical distance of the objects from the hand, first in the horizontal (xy) direction and then in the vertical (z) direction. Complete details for the vicinity FOA mechanism are in figure 4.4(d). Each step of this algorithm scans a square with sides twice the length

[6]Via the workspace model, thus simulating a vision system.

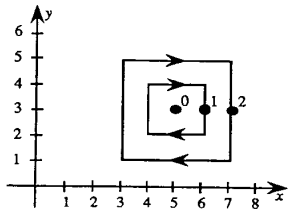


Figure 4.5: Horizontal scanning for the FOA, illustrating horizontal ordering

of the step and center at the robot (x, y) location. It returns any objects within the vicinity distance in the z direction of the hand. Objects providing support are also returned. As figure 4.5 illustrates, if the robot is at (5, 3) then

- max hdist = 0 scans the point (5, 3)
- max hdist = 1 scans the square containing (6, 3), (6, 2), (5, 2), (4, 2), (4, 3), (4, 4), (5, 4), and (6, 4)
- max hdist = 2 scans the square containing (7, 3), (7, 2), (7, 1), (6, 1), (5, 1), (4, 1), (3, 1), (3, 2), (3, 3), (3, 4), (3, 5), (4, 5), (5, 5), (6, 5), (7, 5), and (7, 4)
- *etc*,

stopping as soon as any objects are encountered, but always scanning at least to a max hdist of 1.

Objects in the carry FOA are exposed only as the PMFs are discovered (see section 4.3.4). For example, when grip(*close*) occurs, the grasped object is assumed to be the first in the vicinity FOA so it is moved into the carry FOA; grip(*open*) has the opposite effect.

4.3.3 FOA Point Filter

Many of the points in a teaching sequence are merely "inbetween" points; their existence is irrelevant except that they lie on *one* of many paths which connect the critical positions in the task. ETAR assumes that if the robot is not near an object,

then it is simply in transition between objects. If the vicinity FOA at a point is not empty, then that point is considered important. All other points are disregarded, including those which only have a carry FOA. Objects in the carry move with the robot, effectively becoming another link.

4.3.4 Primitive Motion Functions

The final stage to symbol processing is partitioning the sequence and replacing each partition with a PMF. First this section exposes the Excalibur PMFs, and then discusses partitioning. The PMFs are manipulator dependent, but derived from two general types of motion:

- *precision* — the specific path travelled is important to successful completion of the task. Four precision PMFs are used for the Excalibur:
  1. translate(x, y, z) — straight line motion to (x, y, z)
  2. rotate(*roll*, *pitch*, *yaw*) — smooth orientation change to (*roll*, *pitch*, *yaw*)
  3. transRot(x, y, z, *roll*, *pitch*, *yaw*) — simultaneous translate and rotate, *i.e.*, straight line path over which orientation changes smoothly
  4. grip(*open*/*close*) — gripper opening or closing.
- *imprecision* — the specific path travelled does not matter in the task, provided that the endpoint is reached. The Excalibur implementation has one imprecision PMF:
  moveto(x, y, z, *roll*, *pitch*, *yaw*).

The distance between consecutive position tuples determines the partition on the teach sequence. In proximity to objects, distance is an indication of the teacher's guiding speed because the sampling rate is constant, and so the ratio between distance and speed is constant. Points not near to objects have been eliminated (section 4.3.3) so this speed for accuracy argument does not hold for them. Rather, on

Given:
ϵ, a real number tolerance (0.1)
two points, $p_1 = (x_1, y_1, z_1, roll_1, pitch_1, yaw_1, grip_1)$
$p_2 = (x_2, y_2, z_2, roll_2, pitch_2, yaw_2, grip_2)$
Find:
location change = $|\ (x_1, y_1, z_1) - (x_2, y_2, z_2)\ |$
orientation change = $|\ (roll_1, pitch_1, yaw_1) - (roll_2, pitch_2, yaw_2)\ |$

IF $grip_1 \neq grip_2$ and $grip_2 \neq 0$ then PMF = *grip*.

IF location change > *fast bound* then PMF = *moveto*.
ELSE IF location change > ϵ
IF orientation change > ϵ then PMF = *transRot*.
ELSE IF orientation change $\leq \epsilon$ then PMF = *translate*.
ELSE IF location change $\leq \epsilon$ and orientation change > ϵ
then PMF = *rotate*.
ELSE the PMF is the same as it was previously, *i.e.* the difference in the two points is minimal and disregarded.

(a) PMF between two points

teach sequence	point PMF	PMF sequence
⋮	⋮	⋮
(13.4 -9.3 5.1 1.5 0.0 -2.7 0)	moveto	moveto (13.4 -9.3 5.1 1.5 0.0 -2.7 0)
(13.6 -9.2 4.9 1.7 0.3 -2.5 0)	translate	
(13.4 -8.9 4.1 1.5 0.1 -2.6 0)	translate	
(13.5 -9.3 3.8 1.6 0.1 -2.7 0)	translate	
(13.4 -9.3 3.2 1.4 0.0 -2.7 0)	translate	translate (13.4 -9.3 3.2 1.4 0.0 -2.7 0)
(13.4 -9.3 3.2 1.9 -0.1 -2.1 0)	rotate	
(13.4 -9.3 3.2 1.8 0.0 -1.7 0)	rotate	
(13.4 -9.3 3.2 1.8 0.0 -1.7 0)	rotate	rotate (13.4 -9.3 3.2 1.8 0.0 -1.7 0)
(13.4 -9.3 2.2 1.8 0.0 -1.7 0)	translate	
(13.4 -9.3 2.0 1.8 0.1 -1.7 0)	translate	translate (13.4 -9.3 2.0 1.8 0.1 -1.7 0)
(13.4 -9.3 2.0 1.8 -0.1 -1.7 1)	grip	
(13.4 -9.3 2.0 1.8 0.0 -1.7 1)	grip	grip (13.4 -9.3 2.0 1.8 0.0 -1.7 1)
(13.4 -9.3 2.5 1.9 0.0 -1.7 0)	translate	
(13.4 -9.3 2.9 1.8 0.2 -1.7 0)	translate	
⋮	⋮	⋮

(b) Example (for this case assume that $\epsilon = 0.2$)

Figure 4.6: PMF partitioning

elimination, they are assumed to be imprecision points. ETAR assumes that people tend to slow down as more accuracy is required. This accuracy distance parameter, called *fast bound*, is user dependent, since beginners will move the robot slower than long–time users. More investigation is required here, but the implementation currently fixes *fast bound* at five inches. Figure 4.6(a) is the algorithm for finding the PMF between two points. Briefly, when the distance is greater than *fast bound*, the motion is a *moveto*. Otherwise it is a precision motion: *grip* when there is a change in the gripper, *transRot* when there is a change in both the location and the orientation components of the points, *translate* when only location changes, and *rotate* when only orientation changes. The last PMF and those motion functions that occur just before a change in PMF are kept, thus partitioning the sequence. Arguments to the PMFs are derived from the final point in a partition, as shown in figure 4.6(b).

The FOA is carried along with the PMF, and its numeric arguments, to further generalization where it plays a primary role in reducing the inductive search. Figure 4.7 (repeated from chapter 2) exemplifies the block stacking task at this stage. The result is a sequence of symbolic functions, their arguments, and their FOA. The double–headed arrows should be ignored at this point, since they are introduced in the next section.

4.4 Outer Generalization

The next phase of task construction — outer generalization — matches the PMFs and FOAs. Each example is standardized at the *intra*–example level where an FOA–based search for similarities discovers loops. Between examples, at the *inter*–example level, similar PMFs and FOAs are matched, while differences create branches.

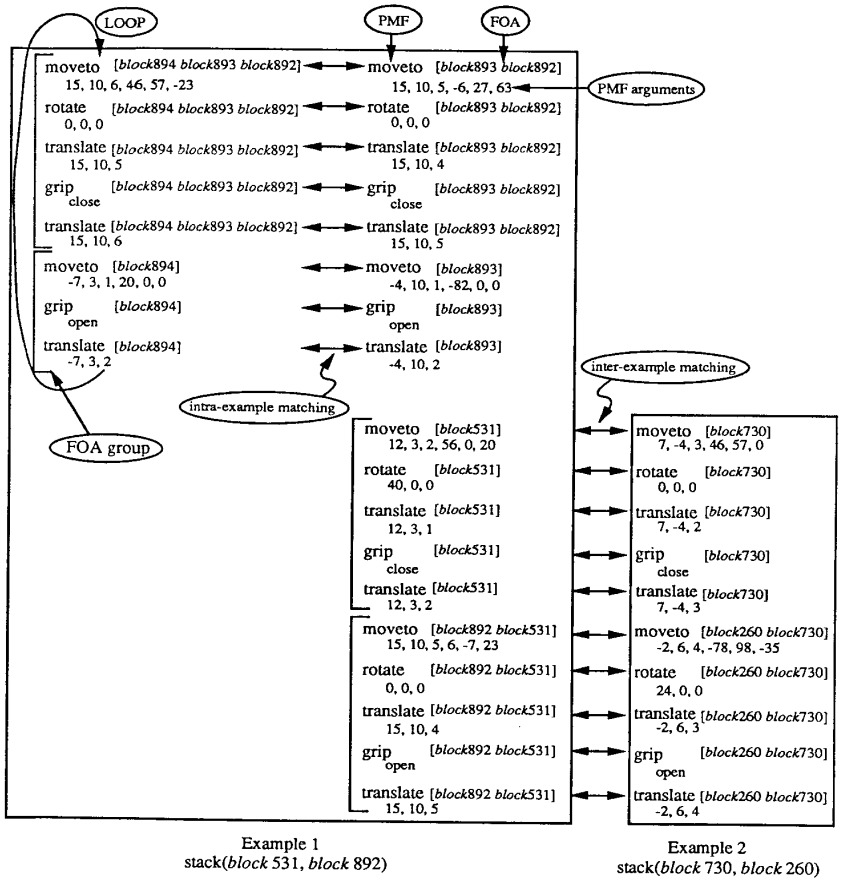


Figure 4.7: The stacking example after symbolic processing and outer generalization

4.4.1 Intra–Example

Three techniques are used to standardize each example: grouping the sequence, combining some PMFs, and searching for loops.

Grouping

Like the conversion into primitive motion functions, matching concentrates on the objects which appear in the FOAs. PMF sequences are partitioned into the largest groups for which the FOA is constant. Figure 4.7 delineates (using brackets) the first and last two groups in the first stacking example.

Rules for Combining Motions

Within each group, subsequences of repetitious motions are combined. For example, a sequence

translate, *rotate*, *translate*, *rotate*, ...

might be one *transRot*. Rules for combining PMFs are manipulator dependent and are represented as production rules. The Excalibur implementation has two:

> If *trSeq* is a sequence of *translates*, *rotates*, and *transRots* containing both a *translate* and a *rotate* component then
>
> *trSeq*, *translate*, *grip* $\longrightarrow$ *transRot*, *translate*, *grip*
> *grip*, *translate*, *trSeq* $\longrightarrow$ *grip*, *translate*, *transRot*.

The purpose of these rules is to account for precision positioning by overshoot and correction during leading. The Excalibur master arm has the control for joint six mounted on the base, separate from the rest of the robot. This joint is used for fine tuning of the horizontal orientation (*roll* about the z-axis), *e.g.* when picking up an object from above. When the robot is close to the position to be attained, the user often moves it a bit closer in location (by using two hands: one on link two and one on link three) then adjusts the orientation (by moving the hand off the robot

model and onto the adjacent servo for joint six). This continues, yielding a sequence of *translates* intermixed with *rotates*, or even *transRots* (since joints two and three do affect orientation). Using the above rules, these sequences are generalized to a simple *transRot* with arguments from the last motion in *trSeq*.

Loop Search

Loops occur on repetition of similar PMFs around different objects sharing a common property. In ETAR, this common property is that the objects are *explicit parameters* of the task, *e.g.* block531 and block892 in stack(block531, block892), or that they are *extra parameters*, *e.g.* block893 in stack(block531, block892). Each example is searched for loops in three steps: determining key groups, discovering potential iterations for a loop, and matching the iterations found.

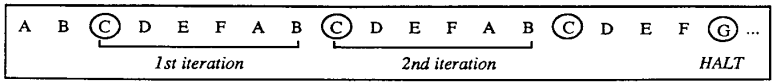
A *key group* contains at least one *key event*, *i.e.*, a motion which causes the robot to begin or stop affecting other objects during its motion.[7] These are the same motions which result in objects entering and leaving the carry FOA: grip(*open*) and grip(*close*) in the Excalibur implementation. Thus, any group which contains the *grip* function is a key group.

Loop iterations occur around key groups. From a key group, a first iteration is determined by traversing through the sequence until another key group with the same key events and FOA type is encountered. Within outer generalization, the FOA type refers to the type of parameters the FOA contains,[8] *i.e.* whether the FOA contains only extra parameters or whether it contains some explicit ones. Further iterations are found by continuing through the sequence, stopping when it is exhausted or a halting condition is met. The halting condition occurs

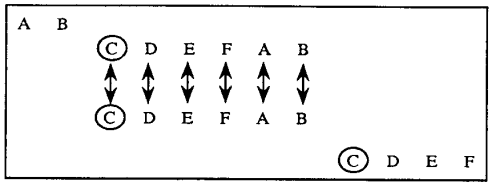
1. on a key group with explicit FOA parameters which are different from the

[7] Similar to NODDY's idea of key event. In NODDY a key event is a unique event in a procedural trace, however, unique events are unlikely in ETAR.

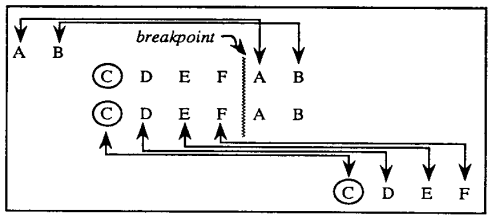
[8] Versus the carry/vicinity distinction in symbol processing.



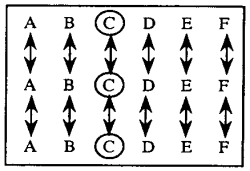
(a) Example sequence
Circles indicate key groups. Alpha characters are FOAs.
G is the only explicit parameter to the task.



(b) Two iterations found and matched



(c) Accounting for non-key starts



(d) The loop

Figure 4.8: Loop search

initial key group's explicit parameters.

2. only after a group without explicit parameters has been passed.

Thus, either the main part of the task is a loop or a task consists of a sequence of subtasks. Each subtask is characterized by movements around one explicit task parameter and any number of extra parameters, and may be a loop, such as clearing the top of a cuboid. Refer to figure 4.8 which contains a simplified example to explain the looping mechanism. It should be clear that the required loop is $\{A\ B\ C\ D\ E\ F\}$. Search for a key group begins at A. C is the first one, so is the base for the loop. Progressing through the sequence, another C is encountered to yield one iteration $\{C\ D\ E\ F\ A\ B\}$. The iteration does not start with A, as was expected, but this will be rectified as the complete loop is generated. A second iteration is quickly found. On search for a third, the G terminates the pursuit for further iterations since it is another key group but now contains an explicit parameter.

Iterations for a loop are matched to form the loop body. Each iteration is treated as one example of the body and the technique for inter–example matching (discussed in the next section) is applied. Figure 4.8(b) shows how the iterations are matched on a strict equality basis. ETAR does not use just equality, but the more complicated match condition based on FOAs and key events mentioned above.

So far the assumption is that loops start on key groups. Perchance they do not. If the iterations match successfully, ETAR attempts to extend the loop into the portion of the sequence directly before and after the iterations. The beginning of one iteration is matched with as much of the sequence after the last iteration as possible. The point in the iteration where matching is first unsuccessful is the *breakpoint* of the loop. In figure 4.8(c), C, D, E, and F are matched, so a breakpoint occurs just before A. If at least one match occurs here, *i.e.*, the breakpoint is not the first element of the iteration, the rest of the iteration is matched to the events just before the first iteration in the sequence. Again in the figure, B and then A are matched so the entire iteration has matched to portions of the sequence directly before and

after. In this case the iterations are flipped about the breakpoint so that the loop starts properly, as shown in part (d) of the figure.

This completes the generation of one loop. Further loops may still exist in the remainder of the sequence, and hence the loop generation algorithm is repeated until all are isolated. Matched loops are later merged by inner induction. Figure 4.7 shows the loop generated in the block stacking example. First, the key group with FOA = [block894 block893 block892] and key event of grip(*close*) is found. Searching for another one with the same key event yields the group with FOA = [block893 block892], so one iteration, shown in the leftmost column, has been determined. It contains the first key group and another key group which has the extra FOA = [block894]. Search continues for more iterations. When the key group with block531 is reached, another iteration is found and the search ends because block531 is a different (from block892) explicit argument to the stacking task.

4.4.2 Inter–Example

Outer generalization ends with inter–example processing where all examples are matched before being fully merged by inner induction. Ideally, each PMF of an example matches to the same PMF in the other examples, *e.g.*

$\vdots$		$\vdots$		$\vdots$
$moveto(*,*,*,*,*,*)$	$\leftrightarrow$	$moveto(*,*,*,*,*,*)$	$\leftrightarrow$	$moveto(*,*,*,*,*,*)$
$rotate(*,*,*)$	$\leftrightarrow$	$rotate(*,*,*)$	$\leftrightarrow$	$rotate(*,*,*)$
$translate(*,*,*)$	$\leftrightarrow$	$translate(*,*,*)$	$\leftrightarrow$	$translate(*,*,*)$
$\vdots$		$\vdots$		$\vdots$
example 1		example 2		example 3.

In practice, inter–example matching is more difficult since tasks may have loops and branches. Matching criteria rooted in FOAs and key events are now developed.

Each example consists of PMFs in FOA groups, possibly with loops. Thus conditions are required, as given in figure 4.9, for matching groups and loops, with the

Loops match only to other loops, and only if their corresponding elements (groups, since loops are not nested) match.

(a) Loop match

Groups match if two conditions hold:
- they contain the same key events.
- they both contain explicit task parameters in their FOA, or neither does. Furthermore, when the number of explicit task parameters is fixed, corresponding ones appear in both FOAs. For example, if stack(block531, block892) is being matched with stack(block730, block260), then groups containing block531 in the FOA cannot match to a group containing block260 in the FOA, since the role of the two blocks is different.

(b) Group match

A loop never matches to a group. However the loop may match to a subsequence starting at the group if the subsequence is a single iteration of the loop [matches as in (a)]. If a match occurs, the subsequence is converted to loop form.

(c) Loop to group match

Figure 4.9: Matching criteria

side effect that the PMFs are also matched. Loops only match to other loops; groups match to other groups with the same key events and with FOAs which play corresponding roles in the task. In other words, if two groups have FOAs which play the same role, *i.e.* occur in the same position in the task specification, and if they contain the same key events, then this is sufficient to assume that the subsidiary PMFs are also the same or can be generalized to the same through omission of unnecessary PMFs. This matching criteria is necessarily strong since the implementation of ETAR has no backtracking.

To understand the matching process, assume that several examples have already been matched to form eg_{gen}. Now a new example, eg_{new}, must be incorporated with the previous ones. ETAR proceeds in parallel through both eg_{gen} and eg_{new}, looking for a match to the first element of either. When a match is discovered, the elements preceding it in the example(s) remain unmatched (matched to *nil*), indicating a branch. The *nil*s are merely placekeepers so the inner induction phase generates

1. If eg_{new} or eg_{gen} is exhausted, match the remaining one to *nil*s and stop.
2. Call the first element of eg_{new} *newCandidate*. Obtain a non–*nil* entry from the first element of eg_{gen} and refer to it as *genCandidate*.[a] Initialize *unMatchedNew* to *newCandidate* and *unMatchedGen* to the element *genCandidate* represents.
3. If *newCandidate* matches with *genCandidate*, then add *newCandidate* to the matched list of groups at the beginning of eg_{gen}. Return to 1 with the remainder of eg_{new} and eg_{gen}.
4. If possible, retrieve both the next element in eg_{new} (call it *testNew*) and a representative of the next element in eg_{gen} (call it *testGen*).
   (a) If *new(gen)Candidate* matches with *testGen(New)*, then all entries of *unMatchedGen(New)* are matched to *nil*s and *new(gen)Candidate* is matched to the element of $eg_{gen(new)}$ which *testGen(New)* represents. Return to 1 with $eg_{new(gen)}$ shortened by one element and $eg_{gen(new)}$ shortened to start after *testGen(New)*.
   (b) Otherwise add *testNew* to *unMatchedNew* and the eg_{gen} element represented by *testGen* to *unMatchedGen*. Return to 4 if either eg_{new} or eg_{gen} has more elements to be tested. If neither does, skip ahead to 5.
5. Neither *newCandidate* nor *genCandidate* match with any element of the opposite example so match both to *nil* and return to 1 with both eg_{new} and eg_{gen} shortened by one.

[a] When attempting to match a single loop to a sequence of groups, *newCandidate* and *genCandidate*, and later *testNew* and *testGen* are extended to contain the single iteration of the loop if it is present in the group sequence.

Figure 4.10: Inter–example generalization algorithm

valid variables. A complete algorithm is in figure 4.10, and the simplified example in figure 4.11 illustrates it. Normally the matching criteria is the one stated earlier, in figure 4.9, but in this example just equality is used to concentrate on the search path. First, in part (1) of figure 4.11, a match is attempted between the heads of both eg_{gen} and eg_{new}. This is impossible, so the search to match one of the heads continues along both examples until the *A* in eg_{new} is found at step (4). The *A* is added into eg_{gen}, along with the preceding unmatched elements in eg_{new}. Matching continues until eg_{new} is exhausted, resulting in the sequence in step (9).

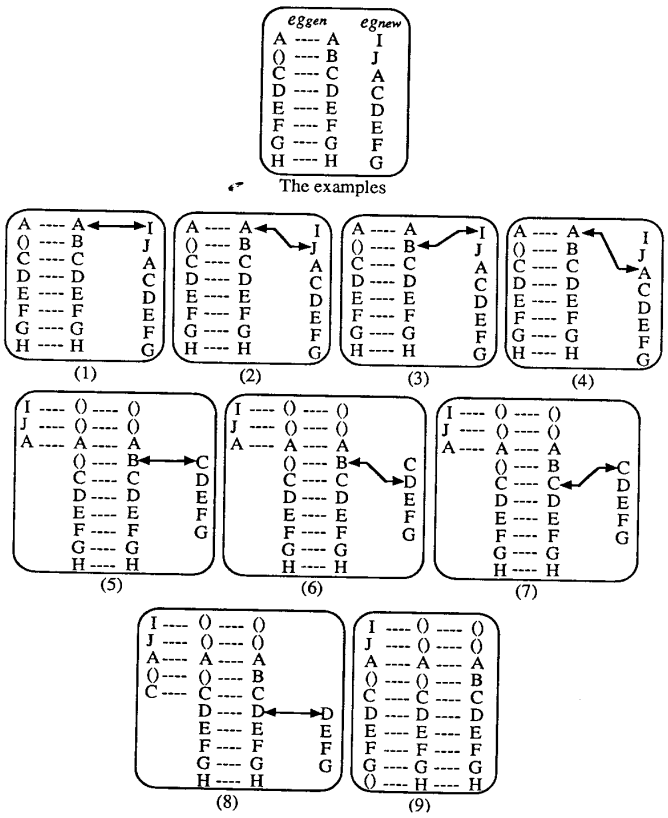


Figure 4.11: Inter–example matching search

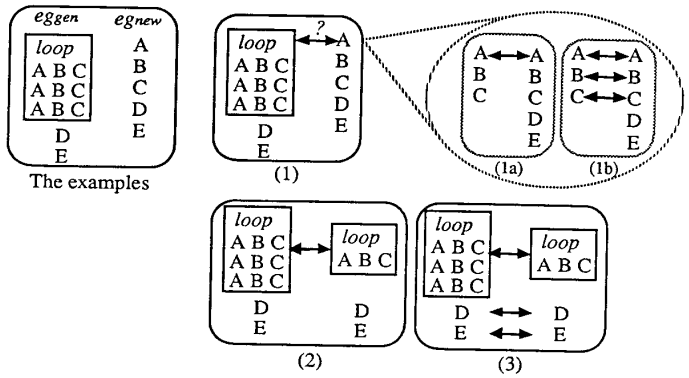


Figure 4.12: Inter–example group to loop match

Figure 4.12 is similar, this time with a loop in one example and none in the other. The search pattern is still the same. First, a match between the heads is attempted [see step (1) in the figure]. ETAR invokes the loop to group matching criteria to determine that the loop in eg_{gen} only matches to the beginning of eg_{new} if eg_{new} contains a single iteration of the loop starting at A. This is discovered true [in steps (1a) and (1b)], so the loop is isolated in eg_{new}, and matched to the head of eg_{gen} as shown in step (2). Matching continues on to the result of step (3).

Now return to the block stacking example. The double–headed arrows between the examples in figure 4.7 show inter–example generalization. First, a match is attempted between the loop of example 1 and the first group (with FOA = [block730]) of example 2. This is impossible, unless example 2 starts with an single iteration of the same loop. But this fails since the second group in example 2 contains block730, an explicit task parameter, and the second group of the loop does not contain an explicit task parameter. The loop is put temporarily on hold, while the next group (with FOA = [block531]) is investigated for a match with the first in example 2. Since a match occurs, the loop is matched to *nil*. Finally, the last group of both

$$
\begin{array}{rl}
foa_1 & motionFn(x_{11},\ x_{12},\ x_{13},\ \ldots,\ \mathbf{x_{1j}},\ \ldots,\ x_{1n}) \\
foa_2 & motionFn(x_{21},\ x_{22},\ x_{23},\ \ldots,\ \mathbf{x_{2j}},\ \ldots,\ x_{2n}) \\
foa_3 & motionFn(x_{31},\ x_{32},\ x_{33},\ \ldots,\ \mathbf{x_{3j}},\ \ldots,\ x_{3n}) \\
\vdots & \qquad\qquad\qquad\quad \vdots \\
foa_m & motionFn(x_{m1},\ x_{m2},\ x_{m3},\ \ldots,\ \mathbf{x_{mj}},\ \ldots,\ x_{mn}) \\
f_j\ :\ input & \longrightarrow output_j
\end{array}
$$

Figure 4.13: Inner induction — each line corresponds to the common PMF occurring at a step of the task which is being learned. A general description, in the form of a function f_j, is sought to describe each argument to the PMF.

examples is matched.

4.5 Inner Induction

Two problems remain at this point of the task construction:

- finding generalized arguments for the matched PMFs
- determining conditionals for loop termination and branches.

Inner induction, so called because it manipulates the inner arguments to the outer level program structure, deals with these problems to produce a complete task description.

4.5.1 General PMF arguments

Suppose that there are m examples of a task and each step of the task contains the same PMF which requires n arguments:

$$motionFn(x_{i1},\ x_{i2},\ x_{i3},\ldots,\ x_{in}).$$

ETAR must determine a functional description for each of the arguments to *motionFn*. This functional description is referred to as an *argument expression* to avoid confusion with the numerous function terms. The only clues available for determining an

argument expression come from the knowledge in the object hierarchy, directed by the FOA. As depicted in figure 4.13, if x_{ij} is the jth argument of the ith example, then the argument expression f_j must be discovered so that

$$\{f_j :\ \text{(slots from object hierarchy)} \longrightarrow x_{ij}\}_{\substack{1\le i\le m \\ 1\le j\le n}}.$$

This problem is twofold:

1. finding the minimum domain of the argument expression, *i.e.*, which slots or aspects related to the FOA affect the outcome.
2. determining the form of the argument expression.

Since the relevant variables are unknown, numerical analysis cannot be used. Instead, symbolic function induction techniques are necessary. Given a set of basis functions, an exhaustive search of the composition space over all known aspects may *eventually* find the required argument expression, but the complexity is overwhelming. Nevertheless, ETAR uses "generate and test" on the composition space but it is controlled by careful composition rules and by limiting the potential aspects upon which the task depends.

Constructing an Argument Expression

Before discussing how the relevant aspects are circumscribed, the algorithm that yields the form of the argument expression is explained. Assume that the relevant aspects constitute an *input* tuple. The corresponding *output* is the argument to generalize in *motionFn*, *i.e.* the x_{ij}. So each example contributes one input–output pair to f_j.

As part of the background knowledge, ETAR has a set of basis functions. The basis functions, under the composition operator, delineate the argument expression space. The implementation basis functions are the standard binary arithmetic operators: multiplication, division, addition, and subtraction. Function induction systematically composes the basis functions into a trial form and tests it with the

The problem: Given a set of input–output tuples, find an argument expression which satisfies them. A set of basis functions is available.

ETAR *'s solution:*

1. If the output is always constant, stop and return the constant value for the argument expression.
2. Initialize the *expression queue* to the list of basis functions. The expression queue will always be a list of functions, where each function contains the function name and its arguments as *uninstantiated variables* or other functions. Members of the expression queue are potential argument expressions. Include the identity function at the beginning of the expression queue.
3. Remove the first element from the expression queue (call it *trial function*) and start the test phase as follows:
   (a) Generate a combination of the input value positions in the input tuple. The length of a combination is the number of uninstantiated arguments in the trial function. For example, if the function requires two arguments and the input tuple has size three, possible combinations would be (1,1), (1,2), (1,3), (2,1), (2,2), (2,3), (3,1), (3,2), and (3,3).
   (b) Evaluate the expression taking its arguments from the input tuples at the positions generated in (a).
   (c) If the result matches the output value for each tuple, then the required argument expression has been found so stop. Otherwise, return to (a) generating a new combination of the input value positions. If all combinations have been exhausted, proceed to step 4.
4. Unless the trial function is the identity, generate all new potential argument expressions by replacing one of the trial function's uninstantiated arguments with a basis function. Place these new expressions[a] at the end of the expression queue. Expansion must be done carefully to avoid adding syntactically redundant expressions. Whenever an uninstantiated argument follows a functional argument at the same level, it is not expanded. For example, if the basis functions contain only one function $h(x, y)$, then the expression queue grows as follows:
   $\{ h(x,y) \}, \{ h(h(x,z),y) \quad h(x,h(y,z)) \}$ ...
   This rule would ensure that only one expression of the form $h(h(x,y),h(z,w))$ is generated at the next level.
5. Return to step 3.

[a]Generation of new expressions is delayed until they are required by the test phase.

Figure 4.14: Function induction

input–output tuples. Since numbers in the robot domain are real, equality is always within a tolerance.[9] Composition proceeds in a top down manner,[10] nesting more functions at each level, and ends when the required argument expression is generated. The complete algorithm is in figure 4.14.

There is no guarantee that a valid argument expression will be discovered, and hence, that the algorithm will halt. A *complexity bound* is introduced to ETAR's function induction to ensure that expressions containing more than a maximum number of basis functions are never generated. Rather than continuing forever, the expression queue eventually empties, halting the induction. These cases result in a '?' as the PMF argument, under the assumption that the value does not matter in the task. Future work might subdivide the domain and generate a function and condition for each subset, but the '?' must still be available as a last resort, since exact values do not always matter.

The general function induction algorithm grows exponentially, as chapter 6 shows. Growth may be decreased by eliminating semantic equivalents. Since all the required functions are numeric, algebraic properties such as symmetry and distributivity explain some duplication and may be used to condense the composition search. Given a set of basis functions, say $\{h(a_1,a_2), g(a_1)\}$ where $h(a_1,a_2)$ is a symmetric function, then the search through the composition space proceeds as in figure 4.15. Notice that if the symmetry was ignored or unknown, then the second stage would have two extra expressions in the composition queue, namely, $\{h(h(a_1,a_2),a_3), h(g(a_1),a_2)\}$. This would magnify throughout the remaining stages, generating many mathematically equivalent functional forms. If the basis functions are always known to be $+$, $-$, $\cdot$, and $/$, then other canonical forms can be used [Falkenhain & Michalski 1986]. Furthermore, algebraic equivalence constrains the combina-

[9]ETAR sets this tolerance at 0.01 units (inches or radians, depending on the variables in the expression).

[10]Some function inducers use bottom up expression construction, *e.g.* [Falkenhain & Michalski 1986, Langley *et al* 1986, Phan 1989].

$1\ \{h(a_1, a_2),\ g(a_1)\}$
$2\ \{g(a_1),\ h(a_1, h(a_2, a_3)),\ h(a_1, g(a_2))\}$
$3\ \{h(a_1, h(a_2, a_3)),\ h(a_1, g(a_2)),\ g(h(a_1, a_2)),\ g(g(a_1))\}$
$4\ \{h(a_1, g(a_2)),\ g(h(a_1, a_2)),\ g(g(a_1)),\ h(h(a_1, a_2), h(a_3, a_4)),$
$h(g(a_1), h(a_2, a_3)),\ h(a_1, h(a_2, h(a_3, a_4))),\ h(a, h(a_2, g(a_3)))\}$
$\vdots \qquad\qquad \vdots$

Figure 4.15: Example of expression queue expansion

tions of inputs used to test function validity. For example, if (height, length) is an incorrect pair of arguments for a symmetric function, then (length, height) need not be tested. The current version of ETAR does not include algebraic constraints.

Finding Relevant Arguments

At the test stage, the function form is evaluated with all combinations of the input, stopping when successful. If there are i input candidates (slots) and a arguments in the expression, then i^a combinations may be tested as each function form is evaluated. In a robot's world, the number of inputs is potentially infinite and must be constrained for the test phase to be feasible. The FOA severely limits the inputs. Only those aspects in the object hierarchy which are common to the FOA associated with *motionFn* are considered. Using the object hierarchy of figure 4.2, if a function is being induced between

- a *motionFn* which has block531 (an instance of cube) as the FOA, and
- a *motionFn* which has block470 (an instance of washer) as the FOA,

then the input is those slots common to both. Thus, height is a possible input, whereas width, length, radius1, and radius2 are not. This minimal set of slots is obtained by climbing up the object hierarchy to the first object which covers[11] all members of the FOA. This object will contain the common slots. If an FOA has more than one member, then input expansion occurs on corresponding elements *i.e.*

[11] Is a supertype of.

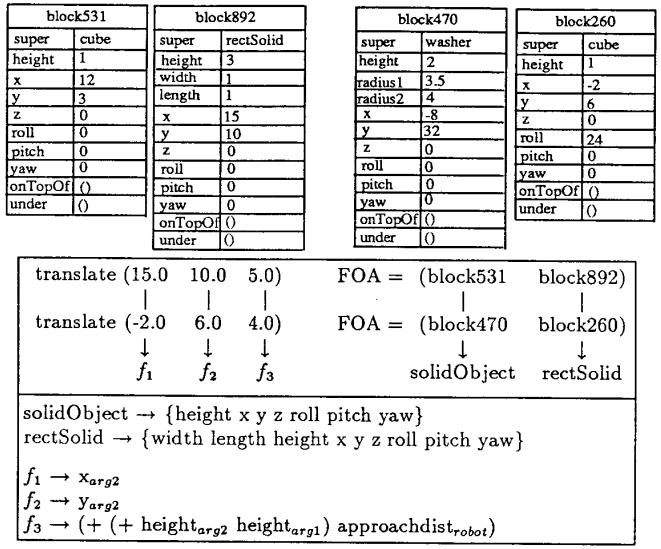


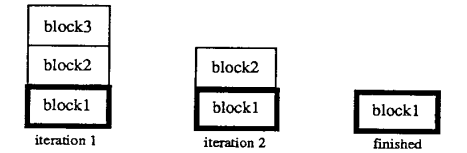
Figure 4.16: Results of ETAR's function induction merging an example

common slots to all first elements, then to all second elements, *etc.* Furthermore, since the basis functions assume a numeric domain, slots whose value is non–numeric are not used.

Figure 4.16 is an example of ETAR merging together a step in a task where the robot has just placed two objects together and is carefully moving to a position just above them. Approachdist = 1″ is a constant in the robot frame. arg_1 and arg_2 are the explicit arguments to the task.

4.5.2 Conditionals for Loops and Branches

Finding branch conditions and loop termination conditions is similar to isolating the relevant variables for function induction, only now qualitative aspects in the FOA



block1	
super	cube
height	1
x	15
y	10
z	2
roll	0
pitch	0
yaw	0
onTopOf	()
under	block2 block3

block1	
super	cube
height	1
x	15
y	10
z	2
roll	0
pitch	0
yaw	0
onTopOf	()
under	block2

block1	
super	cube
height	1
x	15
y	10
z	2
roll	0
pitch	0
yaw	0
onTopOf	()
under	()

Figure 4.17: Search for loop termination condition

are discriminated. Each iteration in a loop is a negative example for the termination condition and termination provides a positive example. Similarly, the first step in a branch is a positive example of that branch condition whereas the alternatives are negative examples. The desired conditional is a difference between the positive example and the negative ones — a difference which Michalski's A^q [Michalski & Chilausky 1980] algorithm is intended to find. A^q is summarized in chapter 3 and further detail can be obtained from the referenced papers. This section concentrates on focussing attention in the relevant places and shows where A^q is used.

The *loop focus* is the intersection of the FOAs at the beginning of each loop iteration and those at the end of the last iteration. In the loop of the stacking example shown again in figure 4.17, the FOA intersection for the two iterations is {block1, block2}. But block2 is moved during the last iteration so the loop focus is only block1. A^q is then applied to the symbolic slots of the loop focus. In the example of figure 4.17, the symbolic aspects of block1 after the loop must be distinguished from the same in each of the iterations. So the loop termination condition is obtained as "under_{block1} = nil". Branching conditions are determined likewise, focussing on

the FOAs at the beginning of the branch.

After this step is complete, so is the general task description. The result for the stacking example is shown again in figure 4.18.[12] Until further examples are provided, this is the final form of the task acquired.

4.6 Compendium

ETAR is a system for programming robots by examples. Users supply these examples by directly leading the robot through the task. Positional feedback is obtained from the robot and is generalized into a task program. The key to the inductive learner is the objects which are being manipulated, *i.e.* the FOA. The FOA controls the search in each stage of the learning algorithm:

1. First, in symbol processing, the hexadecimal joint feedback is transformed into a sequence of PMFs, their arguments, and an associated FOA. Any point in the feedback which is not near an object is disregarded.

2. Then, in outer generalization, PMFs are matched within and between examples to obtain a structure for the program. Repetition in FOAs, along with key events, determines loops within single examples. Matching between the standardized examples is also controlled by the FOA.

3. Finally, in inner induction, expressions are induced for the arguments to matched PMFs and loop/branch conditions. Again, the FOA plays an instrumental role — it limits the possible variables to the common characteristics of the objects in the FOA.

[12]Note that the branch which was generated opposite the loop is subsumed by the loop termination condition. Hence it is eliminated.

stack block1 onto block2
  UNTIL (block2 under nothing) *remove any blocks from* block2
    foa$_1$ = highest object on top of block2
      approach and pick up top block
    moveto(15, 10, $z_{foa1} + height_{foa1} + approachDist_{robot}$, ?, ?, ?)
    rotate(0, 0, 0)
    translate(15, 10, $z_{foa1} + height_{foa1}$)
    grip(*close*)
* translate(15, 10, $z_{foa1} + height_{foa1} + approachDist_{robot}$)
* moveto(?, ?, 1, ?, 0, 0) *move it away*
    grip(*open*)
    translate(?, ?, 2)

  foa$_1$ = block1 *pick up* block1
  moveto(x_{block1}, y_{block1}, $height_{block1} + approachDist_{robot}$, ?, ?, ?)
  rotate($roll_{block1}$, 0, 0)
  translate(x_{block1}, y_{block1}, $height_{block1}$)
  grip(*close*)
* translate(x_{block1}, y_{block1}, $height_{block1} + approachDist_{robot}$)

  foa$_{1,2}$ = block2, block1 *place* block1 *on top of* block2
* moveto(x_{block2}, y_{block2}, $height_{block1} + height_{block2} + approachDist_{robot}$, ?, ?, ?)
* rotate($roll_{block2}$, 0, 0)
* translate(x_{block2}, y_{block2}, $height_{block1} + height_{block2}$)
  grip(*open*)
  translate(x_{block2}, y_{block2}, $height_{block1} + height_{block2} + approachDist_{robot}$)

* indicates an updated object position

? indicates a "don't care" position parameter

Figure 4.18: The stacking task learned

Chapter 5

Low Level Robot Aspects

Before ETAR's learning algorithm could be implemented, the Excalibur's programming level had to be developed to include conversions between joint and world coordinates and straight line motion. These low level system requirements are contained in this chapter. Comprehending these details is not essential to understanding the rest of the thesis, but is necessary for the implementation.

The first part of the chapter explains the *kinematics*, *i.e.* how to correctly transmute between joint and world coordinates. It begins with a summary of Paul's general method, and then adds complex geometric reasoning to obtain original results for the Excalibur. The second part of the chapter clarifies straight line interpolation, focussing on orientation. Orientation interpolation is much more difficult than interpolating location, and the recently revived quaternion method is used in this thesis. A unique introduction to quaternions is presented, ending with the implementation details (from [Heise & MacDonald 1989a]).

5.1 Kinematics

Forward kinematics convert robot joint angles into world coordinates, while inverse kinematics revert this. A full development of the Excalibur's kinematics is included because they have not been previously published[1] and the code supplied with the robot is filled with serious errors.[2] A brief summary of the algebra of linkages starts the discussion, followed by the Excalibur kinematic details.

[1] Except as a department report [Heise & MacDonald 1988].

[2] Though the inverse solution provided by *Robotics Systems International* worked for some (mostly trivial) joint combinations, each joint was incorrect in general.

5.1.1 Overview of the Method

A transformation between bases or coordinate frames defined at each manipulator link is easily accomplished by matrices.[3] [Paul 1981] states a simple method for formulating the matrices. Any manipulator consists of links connected at joints. Movement of each link is either a revolution about (revolute joint) or translation along (prismatic joint) the joint axis. Figure 5.1 shows how each link can be characterized by a length, twist, distance, angle, and basis.[4] With these parameters, transformation from link $n-1$ to link n is comprised of four steps:

1. rotate about z_{n-1} by an angle of θ_n
2. translate along z_{n-1} by distance d_n
3. translate along x_n (was x_{n-1}, now rotated) by length a_n
4. rotate by the twist α_n about x_n.[5]

Expressing this relationship as a product of matrices yields the change of basis matrix, A_n, which takes coordinates from the frame of link n to that of link $n-1$

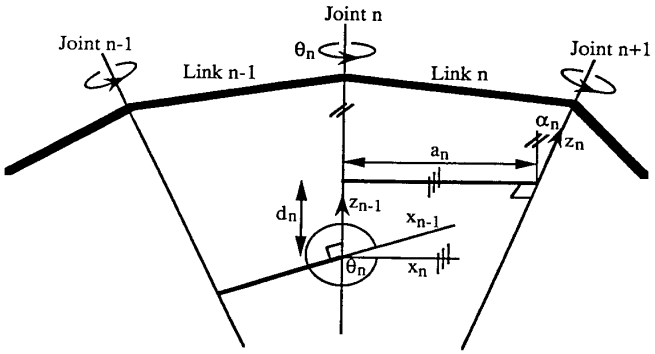
$$
\begin{aligned}
A_n &= Rot(z,\theta_n)Trans(0,0,d_n)Trans(a_n,0,0)Rot(x,\alpha_n) \\
&= \begin{pmatrix} \cos\theta_n & -\sin\theta_n\cos\alpha_n & \sin\theta_n\sin\alpha_n & a_n\cos\theta_n \\ \sin\theta_n & \cos\theta_n\cos\alpha_n & -\cos\theta_n\sin\alpha_n & a_n\sin\theta_n \\ 0 & \sin\alpha_n & \cos\alpha_n & d_n \\ 0 & 0 & 0 & 1 \end{pmatrix}
\end{aligned}
$$

and the overall transform for an n link arm is $T_n = A_1\ A_2\ A_3\ \cdots\ A_n$. ZT_nE is the matrix which transforms end effector coordinates into world coordinates, provided Z is the change of basis from link one to world coordinates and E is the change of basis from the end of the gripper to link n coordinates. ZT_nE can be expressed directly

[3] Change of basis matrices can be found in most elementary algebra books, for example, [Grossman 1987].

[4] All of the Excalibur's joints are revolute. For a discussion of bases at prismatic joints see [Paul 1981].

[5] Angles and distances are expressed in terms of the current basis. For example, the sign of α_n is determined by the direction of x_n.



a_n	length of the common normal connecting the two joint axes
α_n	twist between the two joint axes, measured in a plane perpendicular to the common normal.
d_n	distance separating the two normals along the axis of joint n.
θ_n	angle between two links, measured between the normals in a plane perpendicular to them.
orig_n	intersection of the common normal between joint axes n and $n+1$ and joint axis $n+1$.[a]
z_n	Aligned with the axis of joint $n+1$.
x_n	Aligned with the common normal between joint axis n and joint axis $n+1$.[b]
y_n	$x_n \times z_n$.

[a] Should these axes intersect, the point of intersection is the origin. If the axes are parallel, the origin is so chosen that the joint distance, d_i, is zero to the next defined coordinate frame origin.

[b] If these axes intersect, the x-axis is aligned with $z_{n-1} \times z_n$.

Figure 5.1: Setting up coordinate frames for an arbitrary manipulator (adapted from [Paul 1981])

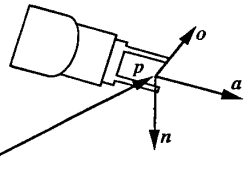


Figure 5.2: Normal, orientation, approach, and position

in world coordinates as a location p and an orientation, illustrated in figure 5.2. The orientation is a normal vector n, an orientation vector o, and an approach vector a, yielding the *noap* matrix:[6]

$$noap = \begin{pmatrix} | & | & | & | \\ n & o & a & p \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{pmatrix}.$$

5.1.2 Excalibur Forward Kinematics

Using the method outlined above, conversion from Excalibur joint coordinates to world coordinates is easily accomplished. Coordinate frames[7] are set up for each link as shown in figure 5.3, resulting in the link parameters of figure 5.4. Substituting these values into A matrices and writing $\cos\theta_n$ as c_n and $\sin\theta_n$ as s_n yields the six matrices of figure 5.5. Two translations, represented in the matrices of figure 5.6, are necessary to finish the conversion:

Z a translation by L_1 along the z-axis to move the world origin to the base of the manipulator

E a translation by L_{56} along the z-axis to the point p at the end of the gripper.

Multiplying all matrices together as $ZA_1A_2A_3A_4A_5A_6E$ forms the *noap* matrix with components shown in figure 5.7, where $c_{23} = \cos(\theta_2 + \theta_3)$ and $s_{23} = \sin(\theta_2 + \theta_3)$.

[6] n, o, and a are easily converted to roll, pitch, and yaw. See [Paul 1981].

[7] There are, of course, many ways of defining these frames, depending on the direction chosen for each of the basis vectors. The frames given here allow expression of length as positive. If the z-axes were pointing down, then length would be negative.

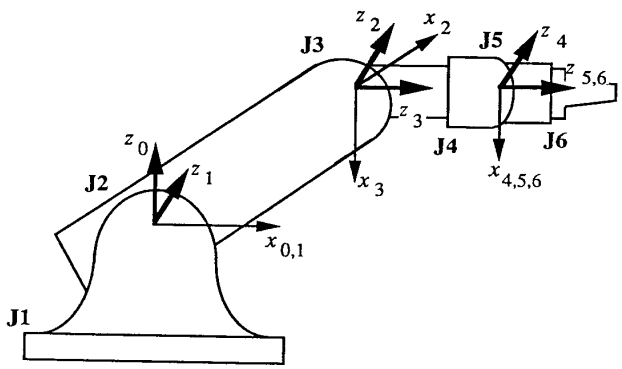


Figure 5.3: Coordinate frames for the Excalibur

link	θ_n	α_n	a_n	d_n	$\cos\alpha_n$	$\sin\alpha_n$
1	θ_1	$-90°$	0	0	0	-1
2	θ_2	$0°$	L_2	0	1	0
3	θ_3	$90°$	0	0	0	1
4	θ_4	$-90°$	0	L_{34}	0	-1
5	θ_5	$90°$	0	0	0	1
6	θ_6	$0°$	0	0	1	0

Figure 5.4: The Excalibur link parameters
$L_{n_1 n_2 \cdots n_k}$ is the combined length of links n_1, n_2, $\cdots$ n_k.

$$A_1 = \begin{pmatrix} c_1 & 0 & -s_1 & 0 \\ s_1 & 0 & c_1 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_2 = \begin{pmatrix} c_2 & -s_2 & 0 & L_2c_2 \\ s_2 & c_2 & 0 & L_2s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_3 = \begin{pmatrix} c_3 & 0 & s_3 & 0 \\ s_3 & 0 & -c_3 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_4 = \begin{pmatrix} c_4 & 0 & -s_4 & 0 \\ s_4 & 0 & c_4 & 0 \\ 0 & -1 & 0 & L_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$A_5 = \begin{pmatrix} c_5 & 0 & s_5 & 0 \\ s_5 & 0 & -c_5 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad A_6 = \begin{pmatrix} c_6 & -s_6 & 0 & 0 \\ s_6 & c_6 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5.5: A matrices for the Excalibur

$$Z = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_1 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad E = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & L_{56} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

Figure 5.6: World origin and tool translation matrices

$$\begin{aligned}
n_x &= c_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] - s_1[s_4c_5c_6 + c_4s_6] \\
n_y &= s_1[c_{23}(c_4c_5c_6 - s_4s_6) - s_{23}s_5c_6] + c_1[s_4c_5c_6 + c_4s_6] \\
n_z &= s_{23}[s_4s_6 - c_4c_5c_6] - c_{23}s_5c_6 \\
\\
o_x &= c_1[s_{23}s_5s_6 - c_{23}(c_4c_5s_6 + s_4c_6)] + s_1[s_4c_5s_6 - c_4c_6] \\
o_y &= s_1[s_{23}s_5s_6 - c_{23}(c_4c_5s_6 + s_4c_6)] - c_1[s_4c_5s_6 - c_4c_6] \\
o_z &= s_{23}[c_4c_5s_6 + s_4c_6] + c_{23}s_5s_6 \\
\\
a_x &= c_1(c_{23}c_4s_5 + s_{23}c_5) - s_1s_4s_5 \\
a_y &= s_1(c_{23}c_4s_5 + s_{23}c_5) + c_1s_4s_5 \\
a_z &= c_{23}c_5 - s_{23}c_4s_5 \\
\\
p_x &= c_1[c_{23}c_4s_5L_{56} + s_{23}(c_5L_{56} + L_{34}) + c_2L_2] - s_1s_4s_5L_{56} \\
p_y &= s_1[c_{23}c_4s_5L_{56} + s_{23}(c_5L_{56} + L_{34}) + c_2L_2] + c_1s_4s_5L_{56} \\
p_z &= -s_{23}c_4s_5L_{56} + c_{23}(c_5L_{56} + L_{34}) - s_2L_2 + L_1
\end{aligned}$$

Figure 5.7: *noap* matrix components

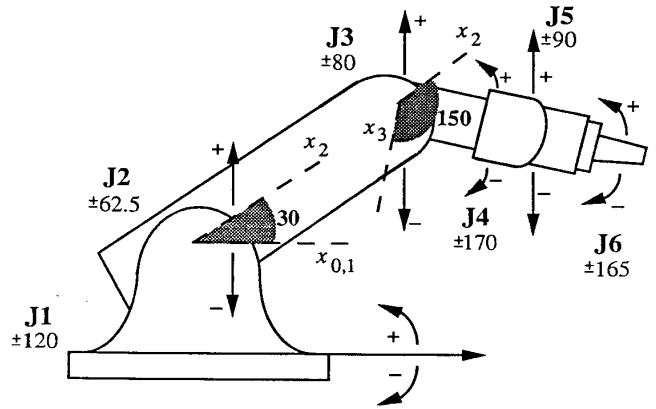


Figure 5.8: Excalibur rest position (all angles read as 0) and joint limits

It is important to notice that the Excalibur robot does not measure its joint angles in exactly the manner suggested above, as shown in figure 5.8. If ρ_n are the angles read from the robot, then the following modifications must be made to the joint values before sending them through the kinematics above.

$$\begin{aligned} \theta_1 &= \rho_1 \quad \theta_2 = -\rho_2 - 30^\circ \quad \theta_3 = -\rho_3 + 150^\circ \\ \theta_4 &= \rho_4 \quad \theta_5 = -\rho_5 \qquad\quad \theta_6 = \rho_6. \end{aligned} \tag{5.1}$$

5.1.3 Excalibur Inverse Kinematics

Inverse kinematics must solve the kinematic equations for all θ_n, thus converting world coordinates to joint angles. This task is more difficult since sine and cosine are not invertible. Algebraic and geometric techniques are combined to find a solution. Many of the angles are determined from properties of triangles, while others are more easily isolated using the matrix equations, premultiplying both sides of $noap = ZA_1A_2\cdots A_nE$ by inverses of the right hand side matrices.

One solution for the joint angles, $\theta_1, \theta_2, \ldots \theta_6$, is now presented. Suppose that

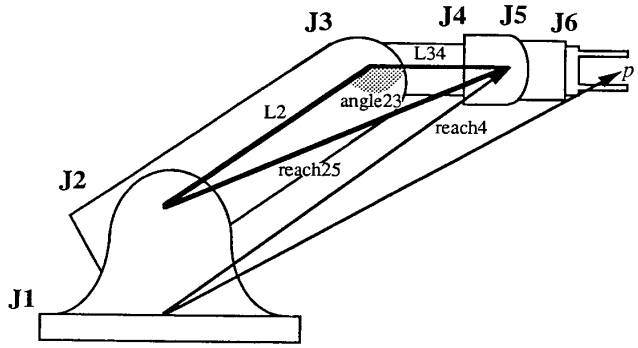


Figure 5.9: Useful *reach* vectors for determining joint angles

the vectors n, o, a, and p are known.

1. Define $reach4$ to be the vector from the base origin to the end of link four, as shown in figure 5.9.

   $$reach4 = (reach4_x, reach4_y, reach4_z) = p - L_{56}a.$$

   This is necessary, since joint one and joint five both *potentially* cause a displacement about the same axis. The vector eliminates the contribution of joint five, enabling the calculation of θ_1. Let $reach4_{xy}$ be the projection of $reach4$ into the x-y plane.

2. Two cases arise when considering the value for θ_1:

   - If the arm is to be placed vertical (*i.e.* $reach4_{xy} = 0$) then it is unnecessary to move the first joint since joints four and six can account for any change required in orientation.
   - Otherwise, $\theta_1 = \text{atan2}(reach4_y, reach4_x)$.[8] If, however, the elbow of the manipulator (joint three) is to be positioned downwards then the solution for joint one is not this easy. Instead, when the above calculation for θ_1

[8] atan2(y,x) = $\tan^{-1}\frac{y}{x}$ where the sign on both components uniquely specifies the quadrant.

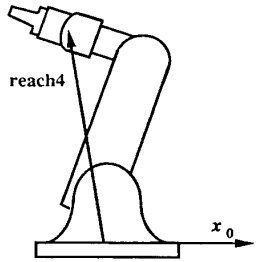


Figure 5.10: The excalibur bent over itself. Elbow is down.

exceeds the manipulator bounds, assume that the robot is bent backwards over the base, as in figure 5.10. Now θ_1 is changed by π. Alternatively, the bounds for joint one may not be exceeded but the solution is still off by π. This condition will only be noticed after solving for the orientation joints (four, five, and six) is found impossible.

3. Let $reach25$ be the length joining joint two to joint five.

$$reach25 = \sqrt{(reach4_{xy})^2 + (reach4_z - L_1)^2}.$$

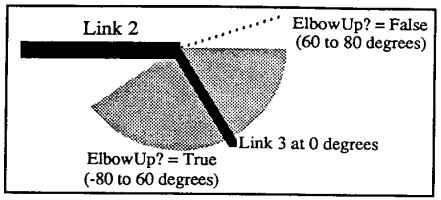
Concentrate on the triangle formed with sides L2, L34, and reach25, visible previously in figure 5.9.

4. Call the angle between links two and three, $angle23$. The cosine of this angle can be determined using the cosine law.[9] This cosine calculation may yield an absolute value greater than one, indicating that the position the robot is required to reach is not within its bounds.

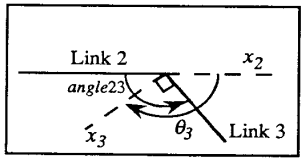
5. Uniquely determining θ_2 and θ_3 is geometrically impossible since their axes are parallel. However, by setting one parameter, $elbowUp?$, this problem is alleviated. Figure 5.11 illustrates the significance of $elbowUp?$.[10] The solution

[9] $\cos\theta = \frac{a^2+b^2-c^2}{2ab}$ where a, b, and c form a triangle and θ is the angle between a and b.

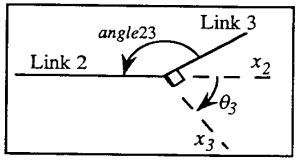
[10] The value of $elbowUp?$ may at times determine whether a solution can be computed. $elbowUp?$ is usually set to true, thus leaving link three less restricted.



(a) Range of link 3



(b) ElbowUp? = True



(c) ElbowUp? = False

Figure 5.11: Finding the angle at link 3

for θ_3 is then

$$\theta_3 = \begin{cases} 270° - angle23 & \text{if } elbowUp? = \text{true}^{11} \\ angle23 - 90° & \text{otherwise.}^{12} \end{cases}$$

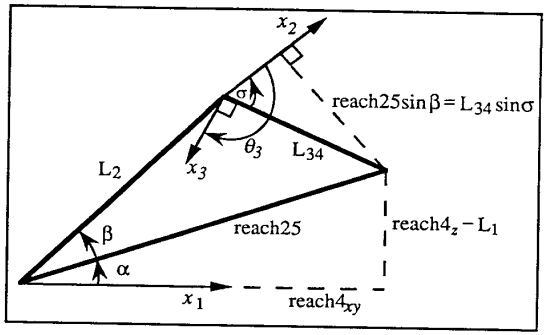
6. The next task is to solve for θ_2 by summing two angles: α and β, as shown in figure 5.12. Depending on the results for θ_1 and θ_3, the arm configuration varies and the figure depicts three of the possibilities. α and β are calculated the same in each case:

   - $\alpha = \text{atan2}\,(reach4_z - L_1,\ reach4_{xy})$
   - $\beta = \text{atan2}\,(\sin\beta,\ \cos\beta)$ where $\cos\beta$ is determined using the cosine law on the triangle $reach25$-L_2-L_{34} and $\sin\beta$ is determined using similar right triangles.

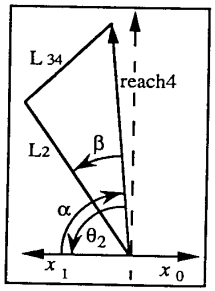
$$\begin{aligned} \sin\beta &= \frac{L_{34}\sin\sigma}{reach25} \\ &= \frac{L_{34}\sin(\theta_3 - 90°)}{reach25}. \end{aligned}$$

[11] This formula is more easily visualized as $(180° - angle23) + 90°$.

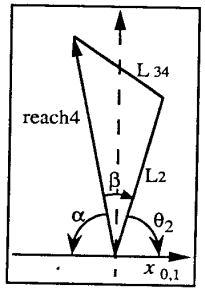
[12] Determined using complementary angles about the lines formed by link 2 and link 3.



(a) Elbow is up



(b) Elbow is down swing is same direction as $reach4_x$



(c) Elbow is down swing is opposite direction as $reach4_x$

Figure 5.12: Summing α and β to determine θ_2

When the elbow is up as in part (a) of figure 5.12, $\theta_2 = -(\alpha + \beta)$. Otherwise the elbow is down and the solution depends on whether the swing (joint one) axis is in the same direction as $(reach4_x,\ reach4_y)$. If so, *i.e.* the cosine of θ_1 has the same sign as $reach4_x$, then once again $\theta_2 = -(\alpha + \beta)$, arising in part (b) of the figure. In the remaining case, (c) of the figure, $\theta_2 = \alpha - \beta - \pi$.

7. Adding θ_2 and θ_3 gives θ_{23}.

   Solving for the remaining three joints is difficult and resorting to algebra is helpful. These joints are primarily concerned with the orientation of the end effector, whereas the first three were concerned with attaining the desired position.

8. The cosine of angle five is isolated in the equation $A_3^{-1}A_2^{-1}A_1^{-1}Z^{-1}(noap) = A_4A_5A_6E$:

$$\begin{pmatrix} c_1c_{23} & s_1c_{23} & -s_{23} & L_1s_{23} - L_2c_3 \\ -s_1 & c_1 & 0 & 0 \\ c_1s_{23} & s_1s_{23} & c_{23} & L_1c_{23} - L_2s_3 \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} | & | & | & | \\ n & o & a & p \\ | & | & | & | \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} f_{31}(n) & f_{31}(o) & f_{31}(a) & f_{31}(p) + L_1s_{23} - L_2c_3 \\ f_{32}(n) & f_{32}(o) & f_{32}(a) & f_{32}(p) \\ f_{33}(n) & f_{33}(o) & f_{33}(a) & f_{33}(p) + L_1c_{23} - L_2s_3 \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

$$= \begin{pmatrix} c_4c_5c_6 - s_4s_6 & -c_4c_5s_6 - s_4c_6 & c_4s_5 & L_{56}c_4s_5 \\ s_4c_5c_6 + c_4s_6 & -s_4c_5s_6 + c_4c_6 & s_4s_5 & L_{56}s_4s_5 \\ -s_5c_6 & s_5s_6 & c_5 & L_{56}c_5 + L_{34} \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

where[13]

$$\begin{aligned} f_{31}(v) &= (c_1c_{23})v_1 + (s_1c_{23})v_2 - s_{23}v_3 \\ f_{32}(v) &= -s_1v_1 + c_1v_2 \\ f_{33}(v) &= (c_1s_{23})v_1 + (s_1s_{23})v_2 + c_{23}v_3. \end{aligned}$$

9. Comparing the value in position 3-3 of both sides of this equation gives

$$\cos\theta_5 = f_{33}(a).$$

[13] f_{ij} refers to the function resulting from the matrix equation $A_i^{-1}A_{i-1}^{-1}\cdots A_1^{-1}T_n = A_{i+1}A_{i+2}\cdots A_n$ in the jth row [Paul 1981].

10. In the trivial case ($c_5 = 1$), $\theta_5 = 0$ since $-90° \leq \theta_5 \leq 90°$. This implies that the axes of rotation for joints four and six are parallel, hence the rotation should be split between the two. The previous matrix equation provides the necessary value:

$$\begin{aligned}\frac{f_{32}(n)}{f_{31}(n)} &= \frac{s_4c_5c_6 + c_4s_6}{c_4c_5c_6 - s_4s_6} \\ &= \frac{s_4c_6 + c_4s_6}{c_4c_6 - s_4s_6} \quad \text{since } c_5 = 1 \\ &= \frac{\sin(\theta_4 + \theta_6)}{\cos(\theta_4 + \theta_6)}.\end{aligned}$$

Thus $\theta_4 + \theta_6 = \text{atan}(f_{32}(n),\ f_{31}(n))$. Assigning actual values to θ_4 and θ_6 can be done in many ways. Whenever possible it is a good idea to ensure that the amount of motion occurring in both joints is approximately the same. This is accomplished by finding the difference between the new required value for $\theta_4 + \theta_6$ and the old value, adding half to each joint.

11. If $\theta_5 \neq 0$ it is generally impossible to find a unique solution for the last three joints. This interaction is different from that between joints two and three, where a parameter was allocated to aid the solution. There each solution resulted in a configuration of the manipulator which *looked* different. In this case the different solutions result in the manipulator looking as if it is in the same position. Hence the actual solution found is not important in the function of the manipulator.[14] The strategy used is to assume that θ_5 is a positive angle, then calculate θ_4 and θ_6 and ensure that they fall within the limits of the manipulator. If they do not, then assume that θ_5 is negative. This means that θ_4 and θ_6 must be moved to the opposite quadrant by adding or subtracting 180°.

    - Let $0 < \theta_5 \leq 90°$, where θ_5 is determined by an inverse cosine. Calculate θ_4 and θ_6, from the matrix equation, checking that they fall within the manipulator bounds.

[14] That is, provided that joint limitations do not come into play.

Agree that
roll = 180° *pitch* = 0° *yaw* = 180°
is the same orientation as
roll = 0° *pitch* = 180° *yaw* = 0°.

Interpolate from
roll = 180° *pitch* = 0° *yaw* = 180°
to
roll = 0° *pitch* = 175° *yaw* = 0°.

Result (pictured to the right)
1) *roll* = 180° *pitch* = 0° *yaw* = 180°
2) *roll* = 144° *pitch* = 35° *yaw* = 144°
3) *roll* = 108° *pitch* = 70° *yaw* = 108°
4) *roll* = 72° *pitch* = 105° *yaw* = 72°
5) *roll* = 36° *pitch* = 140° *yaw* = 36°
6) *roll* = 0° *pitch* = 175° *yaw* = 0°.

What a mess, considering the two orientations were close together to start with!

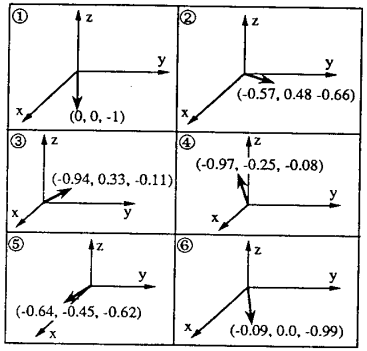


Figure 5.13: Linearly interpolating *roll*, *pitch*, and *yaw*

$$\begin{aligned}\theta_4 &= \text{atan2}\,(f_{32}(a), f_{31}(a)) \\ \theta_6 &= \text{atan2}\,(f_{33}(o), -f_{33}(n))\end{aligned}$$

- Otherwise θ_5 is negative, so θ_4 and θ_6 are in the opposite quadrant from that calculated in the previous step.

12. Make sure that all joints calculated fall within the mechanical limitations of the robot.

Once again, since the Excalibur robot measures joints from different starting positions it is necessary to convert the angles resulting from this algorithm by inverting equation 5.1 on page 90.

5.2 Motion Interpolation

Now that kinematics are complete, discussion turns to the implementation of

the path–controlled PMFs: *translate*, *rotate*, and *transRot*.[15] The world position of the robot end effector is expressed as a location (x, y, z) and an orientation (*roll*, *pitch*, *yaw*) which must be interpolated to yield a smooth trajectory. Location can be trivially interpolated by a straight line function. Difficulty lies in interpolating the orientation. It is not enough to linearly change each of the angles (*roll*, *pitch*, *yaw*), since this results in an uneven overall motion. Figure 5.13 illustrates the problem — it is not obvious when two orientations are close together and hence the interpolated path may jump far from the goal before converging. Quaternions avoid this problem, offering the following advantages:

1. explicit representation of any orientation as the cosine of an angle and the axis of revolution
2. unique representation of an orientation, provided the angle θ satisfies $0° \leq \theta < 180°$ (which is always possible)
3. smooth interpolation between orientations is easily achieved
4. cost for quaternion manipulation is considerably less than the matrix alternative.

This section defines quaternions and their operations. Then it shows how quaternion multiplication causes vector rotation. Straight–line interpolation, using quaternions for orientation, follows. Finally, the implementation for the Excalibur is discussed. This includes the conversion process between robot joint angles and quaternions.

5.2.1 What is a Quaternion?

A quaternion [Hamilton 1969, Kelland & Tait 1904] is simply a four–vector: it inherits all vector properties and operations, including dot product, scalar multiplication, addition and norm. Where quaternions are special is in the definition of

[15] *moveto* is a robot controller command. *grip* combines the OPEN and CLOSE controller commands.

A Quaternion $q = S + X\mathbf{i} + Y\mathbf{j} + Z\mathbf{k} = [S, \bar{w}]$ where $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$	
addition	$q_1 + q_2 = [S_1, \bar{w_1}] + [S_2, \bar{w_2}]$ $= [S_1 + S_2, \bar{w_1} + \bar{w_2}]$
additive identity	$0 = [0, \bar{0}]$
scalar multiplication	$\kappa q = [\kappa S, \kappa \bar{w}]$
multiplication	$q_1 q_2 = [S_1, \bar{w_1}][S_2, \bar{w_2}]$ $= [S_1 S_2 - \bar{w_1} \cdot \bar{w_2}, S_1 \bar{w_2} + S_2 \bar{w_1} + \bar{w_1} \times \bar{w_2}]$
multiplicative identity	$1 = [1, \bar{0}]$
multiplicative inverse	$q^{-1} = [\frac{S}{\zeta}, \frac{-\bar{w}}{\zeta}]$ where $\zeta = \parallel q \parallel$

Figure 5.14: Quaternion operations

multiplication, which is traditional algebraic multiplication with the property that $\mathbf{i}^2 = \mathbf{j}^2 = \mathbf{k}^2 = \mathbf{ijk} = -1$, turning quaternions into a non–abelian division ring [Herstein 1975]. Figure 5.14 summarizes the common quaternion operations and [Heise & MacDonald 1989a] provides further detail and motivation.

5.2.2 Quaternion Multiplication and Three–Space

An important use for quaternions is vector rotation. Any three–vector[16] $\bar{v}$ can be mapped into four space as $v = [0, \bar{v}]$ and treated as a quaternion. When such vectors are multiplied by quaternions, rotation and scaling may occur. By considering a special subset of quaternions, the unit quaternions, only rotation happens since vector norm is preserved. A rotation of $\bar{v}$ by θ around the unit axis $\bar{u}$ is given by qvq^{-1} where

$$q = [\cos\frac{\theta}{2}, (\sin\frac{\theta}{2})\bar{u}].$$

[16] The remainder of this chapter uses bar notation to indicate that the vector comes from three–space, italic capitals to distinguish scalars, and italic lower case for quaternions.

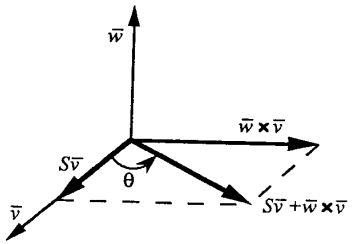


Figure 5.15: Effect of a quaternion on a perpendicular vector

An explanation justifying this expression proceeds below and gives intuitive insight into quaternion rotation. Any vector $\bar{v}$ can be decomposed into vectors perpendicular and parallel to any other vector $\bar{w}$. It is informative to determine the effect of quaternion multiplication on a three–vector by examining rotation about a perpendicular axis and about a parallel axis. Combining the two cases results in an understanding of general quaternion rotation.

Case I: Perpendicular

Suppose that the vector $\bar{v}$ is perpendicular to the vector portion of a unit quaternion $q = [S, \bar{w}]$, *i.e.* $\bar{v} \perp \bar{w}$. Multiplying on the left by q yields

$$\begin{aligned} qv &= [S, \bar{w}][0, \bar{v}] \\ &= [-\bar{w}\cdot\bar{v},\ S\bar{v} + \bar{w}\times\bar{v}] \\ &= [0,\ S\bar{v} + \bar{w}\times\bar{v}]. \end{aligned}$$

As the scalar portion is zero, the result represents another three–vector which is the original $\bar{v}$ rotated about $\bar{w}$, as illustrated in figure 5.15. To ascertain what this new vector is, it is necessary to determine its length and the angle through which it has rotated.

Using Pythagorus' Theorem, the squared length of the new vector is:

$$\| S\bar{v} + \bar{w}\times\bar{v} \|^2 = \| S\bar{v} \|^2 + \| \bar{w}\times\bar{v} \|^2$$

$$\begin{aligned} &= S^2 \| \bar{v} \|^2 + \| \bar{w} \|^2 \| \bar{v} \|^2 \\ &\qquad \text{from Lagrange's Identity} \\ &= (S^2 + \| \bar{w} \|^2) \| \bar{v} \|^2 \\ &= \| \bar{v} \|^2 \qquad \text{since } S^2 + \| \bar{w} \|^2 = \| q \| = 1. \end{aligned}$$

The length of the resulting vector is the same as the length of the original vector. It has, however, been rotated through an angle, θ, about the axis $\bar{w}$. This angle arises from the right triangle formed between $S\bar{v}$ and $S\bar{v} + \bar{w}\times\bar{v}$:

$$\cos\theta = \frac{\| S\bar{v} \|}{\| \bar{v} \|} = S.$$

The angle of rotation determines the first element or scalar portion of the quaternion, while the axis of rotation determines the vector portion. This vector portion must be chosen to ensure that the norm of the quaternion is one.

$$\begin{aligned} S^2 + \| \bar{w} \|^2 &= 1 \\ \cos^2\theta + \| \bar{w} \|^2 &= 1 \\ \| \bar{w} \| &= \sin\theta. \end{aligned}$$

Thus, $\bar{w} = (\sin\theta)\cdot$(unit axis of the rotation). Multiplying by a unit quaternion causes rotation of a vector that is perpendicular to the vector portion of the quaternion. The parallel case must now be considered, after which it shall be necessary to return to this perpendicular case.

Case II: Parallel

Now assume that the vector $\bar{v}$ is a scalar multiple of the vector portion of q, *i.e.* $A\bar{v} = \bar{w}$ for some scalar A. Rotation of a vector which lies on the axis of rotation should leave it unaltered. To determine if this happens with quaternions multiply by q:

$$\begin{aligned} qv &= [S, \bar{w}][0, \bar{v}] \\ &= [-\bar{w}\cdot\bar{v},\ S\bar{v} + \bar{w}\times\bar{v}] \\ &= [-\bar{w}\cdot\bar{v},\ S\bar{v}]. \end{aligned}$$

As the scalar portion of this result is not zero, qv cannot be interpreted as purely a rotation of v. To obtain a quaternion with a null scalar, examine qvq^{-1}.

$$\begin{aligned} qvq^{-1} &= [-\bar{w}\cdot\bar{v},\ S\bar{v}][S,\ -\bar{w}] \\ &= [-S\bar{w}\cdot\bar{v}+S\bar{v}\cdot\bar{w},\ (\bar{w}\cdot\bar{v})\bar{w}+S^2\bar{v}-S\bar{v}\times\bar{w}] \\ &= [0,\ S^2\bar{v}+(\bar{w}\cdot\bar{v})\bar{w}]. \end{aligned}$$

Further simplification of the vector portion confirms the expected identity.

$$\begin{aligned} S^2\bar{v}+(\bar{w}\cdot\bar{v})\bar{w} &= S^2\bar{v}+(A\bar{v}\cdot\bar{v})A\bar{v} \\ &= S^2\bar{v}+(A\bar{v}\cdot A\bar{v})\bar{v} \\ &= (S^2+\parallel\bar{w}\parallel^2)\bar{v} \\ &= \bar{v}. \end{aligned}$$

It is essential to return to case I and determine how qvq^{-1} fares in the perpendicular case. Look at vq^{-1}:

$$\begin{aligned} vq^{-1} &= [0,\bar{v}][S,-\bar{w}] \\ &= [\bar{v}\cdot\bar{w}, S\bar{v}-\bar{v}\times\bar{w}] \\ &= [0, S\bar{v}+\bar{w}\times\bar{v}] \\ &= qv. \end{aligned}$$

In the perpendicular case, qvq^{-1} rotates the vector $\bar{v}$ twice as far about the same axis as qv. Thus qvq^{-1} is a general method for quaternion rotation. The next section summarizes this discussion, yielding a general formula for quaternion rotation.

General Quaternion Rotations

Suppose that $q=[S,\bar{w}]$ is a unit quaternion, where $S=\cos\alpha$ and $\bar{w}=(\sin\alpha)\cdot$(unit axis of rotation). Any vector $\bar{v}$ can be written as the sum of two vectors:

1. a part perpendicular to $\bar{w}$ and
2. a part parallel to $\bar{w}$.[17]

Combining cases I and II indicates that qvq^{-1} is a rotation which leaves the portion of $\bar{v}$ parallel to the axis alone and rotates the perpendicular part by 2α. This is

[17] See any first year algebra textbook, *e.g.* [Anton 1981].

Rotation of vector $\bar{v}$ by θ about the unit axis $\bar{u}$ is given by the vector portion of

$$qvq^{-1}=[0,\bar{v}+2S(\bar{w}\times\bar{v})+2\bar{w}\times(\bar{w}\times\bar{v})]$$

where

$$q=[S,\bar{w}]=[\cos\frac{\theta}{2},(\sin\frac{\theta}{2})\bar{u}]$$

Figure 5.16: Quaternion rotation (see [Funda 1988] for this simplified formula.)

summarized in figure 5.16. The crux is to understand that quaternions, like matrices, may be used for vector rotation, and hence are an alternative representation for orientation.

5.2.3 Interpolation

Moving the "hot spot"[18] of a robot smoothly on a straight line is simple linear interpolation of Cartesian three–space. But what if the orientation of the hand must change at the same time? In some cases, this may not be an issue since orientation can be changed once — at the end of the move. Other times, such drastic motions are intolerable anywhere along the path. It is essential that orientation change evenly throughout. Any general linear interpolation (*e.g. transRot*) would combine both location and orientation interpolation.

Location (Linear) Interpolation: *translate*

Moving from a location p_1 to a new location p_2, as t goes from one to zero, is accomplished by taking a fraction of the difference between the two points [Brady 1986, Taylor 1986].

$$new\_location(t)=t(p_1)+(1-t)p_2=p_2-t(p_2-p_1). \tag{5.2}$$

[18] The point directly between the two fingers of the gripper.

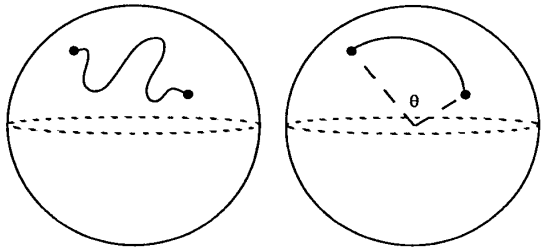


Figure 5.17: Interpolating on the sphere

Orientation (Spherical) Interpolation: *rotate*

A natural way to handle orientation is through unit quaternions, where each orientation is represented as the cosine of an angle and an axis. Just as in three–space smooth motion occurs on a straight line — the shortest path between two vectors — the four space interpolation path must travel the shortest path between two quaternions. Since orientations lie on the unit 4–D sphere, interpolation means traversing the "arc" joining two unit quaternions. Interpolating roll, pitch, and yaw did not work because the path was jumping all over the unit 4–D sphere, as suggested in the 3–D interpretation of figure 5.17(a). Shown in part (b) of the figure is the desired, shortest path. The equation for this arc moving from q_1 to q_2 as t goes from one down to zero is:

$$new\_orientation(t) = \frac{\sin(t\theta)}{\sin\theta}q_1 + \frac{\sin[(1-t)\theta]}{\sin\theta}q_2 \tag{5.3}$$

where $\cos\theta = q_1 \cdot q_2$.[19] The equation does what is required — except when $\sin\theta = 0$, *i.e.* $\theta = 0, \pi, -\pi$. When θ is close to zero, the arc between the two quaternions looks much like a straight line. Linear interpolation of four–space should be done, using formula 5.2 above. If $\theta = \pm\pi$ then $q_1 = -q_2$ and both quaternions represent the same orientation. In fact, restricting orientations to one half of the 4–D sphere ensures that this will not happen and that all orientations have a unique representation.

[19]Suggested in [Shoemake 1985] and proven in [Heise & MacDonald 1989a].

roll, pitch, yaw	quaternion
roll = 180° *pitch* = 0° *yaw* = 180°	(0.000000 0.0 1.000000 0.0)
roll = 180° *pitch* = 1° *yaw* = 180°	(0.008728 0.0 0.999962 0.0)
roll = 180° *pitch* = 2° *yaw* = 180°	(0.017460 0.0 0.999846 0.0)
roll = 180° *pitch* = 3° *yaw* = 180°	(0.026190 0.0 0.999657 0.0)
roll = 180° *pitch* = 4° *yaw* = 180°	(0.034911 0.0 0.999390 0.0)
roll = 180° *pitch* = 5° *yaw* = 180°	(0.043619 0.0 0.999048 0.0)

Figure 5.18: Results of implementation of quaternion interpolation on problem posed in figure 5.13

Figure 5.18 returns to the interpolation problem posed at the beginning of this section. Now quaternion interpolation is employed and the corresponding *roll*, *pitch*, and *yaw* values are given (the conversion is discussed below). This result is superior to that of the *roll*, *pitch*, *yaw* interpolation of figure 5.13 — there is far less motion, instead, movement is always towards the goal.

[Taylor 1986] provides a brief description of an alternate formulation for quaternion interpolation. He determines a quaternion q_{int} which transforms q_1 into q_2 through composition, *i.e.* $q_{int} = q_1^{-1}q_2$. From q_{int} the angle θ and axis of rotation $\bar{u}$ are determined, so that quaternion interpolation from q_1 to q_2 as t goes from one to zero is

$$q_1[\cos\frac{(1-t)\theta}{2},\ \sin\frac{(1-t)\theta}{2}\bar{u}].$$

The computational requirements of this algorithm exceed that of the arc formula in equation 5.3.

Comparisons between quaternions and matrices in rotational tasks are given in figure 5.19. Notice that the complexity in using quaternions is lower than in using matrices for all tasks except vector rotation. Even here, quaternions are generally the preferred representation since setting up the matrix is more complex than determining the quaternion for rotation.

Operation[a]	Quaternions	Matrices
Rotating a Vector	15 M, 12 A	9 M, 6 A
Composition of Rotations	16 M, 12 A	24 M, 15 A
Setting up the rotation from an angle and a unit axis	4 M, 1 A, 1 Sqrt, 1 Trig	22 M, 11 A, 1 Sqrt, 1 Trig
Extracting angle and axis from rotation	4 M, 1 A, 1 Trig, 1 Sqrt	10 M, 16 A, 2 Sqrt, 1 Trig
Interpolation — finding the next rotational knot point	8 M, 4 A, 2 Trig	30 M, 15 A

[a]References to the formulae used in calculating the operation counts appear in appendix A

Figure 5.19: Operation counts for rotation tasks

5.2.4 Implementation on the Excalibur

The kinematics described earlier manipulate world positions (x, y, z, *roll*, *pitch*, *yaw*), so a conversion to and from quaternions is necessary — a gap which this section bridges. Figure 5.20 diagrams the steps involved in robot straight-line motion — starting with joint angles, converting to a location and a quaternion, performing the interpolation, and finally, converting back to joint angles to move the robot. Ideally, the two extra steps necessary to convert between conventional orientation and quaternions would be eliminated, resulting in a direct path between joint angles and quaternions. More research in quaternion kinematics of general manipulators is necessary, but [Funda 1988] provides a good example in the application of quaternions to solve the inverse kinematics of a Puma robot arm. Following the conversion between the three equivalent forms of orientation, considerations for implementing straight-line motion on a robot are presented.

Roll, Pitch, and Yaw to Quaternion

A series of rotations becomes a quaternion by converting each individual rotation into a quaternion and multiplying them together in the proper order. Expressing

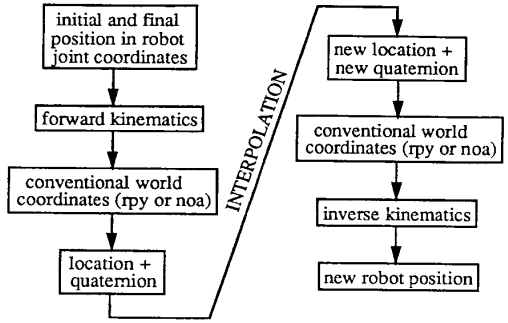


Figure 5.20: Robot motion interpolation

each of *roll*, *pitch*, and *yaw* as a quaternion yields:

$$\begin{aligned} q_{roll} &= [\cos\frac{roll}{2},\ (0,0,\sin\frac{roll}{2})] \\ q_{pitch} &= [\cos\frac{pitch}{2},\ (0,\sin\frac{pitch}{2},0)] \\ q_{yaw} &= [\cos\frac{yaw}{2},\ (\sin\frac{yaw}{2},0,0)]. \end{aligned}$$

Multiplying these together as $q_{roll}q_{pitch}q_{yaw} = [S,X,Y,Z]$ gives the desired quaternion with[20]

$$\begin{aligned} S &= \cos\frac{yaw}{2}\cos\frac{pitch}{2}\cos\frac{roll}{2} + \sin\frac{yaw}{2}\sin\frac{pitch}{2}\sin\frac{roll}{2} \\ X &= \sin\frac{yaw}{2}\cos\frac{pitch}{2}\cos\frac{roll}{2} - \cos\frac{yaw}{2}\sin\frac{pitch}{2}\sin\frac{roll}{2} \\ Y &= \cos\frac{yaw}{2}\sin\frac{pitch}{2}\cos\frac{roll}{2} + \sin\frac{yaw}{2}\cos\frac{pitch}{2}\sin\frac{roll}{2} \\ Z &= \cos\frac{yaw}{2}\cos\frac{pitch}{2}\sin\frac{roll}{2} - \sin\frac{yaw}{2}\sin\frac{pitch}{2}\cos\frac{roll}{2}. \end{aligned}$$

Quaternion to Roll, Pitch, and Yaw

Conversion in the other direction is much more difficult, since roll, pitch, and yaw angles are not unique. Inverting the previous equations to solve for roll, pitch, and

[20]Different from Shoemake [Shoemake 1985], since he treats quaternion rotation as $q^{-1}vq$ and uses a left-handed coordinate system.

yaw is practically impossible, so another method must be sought. If a transformation is represented in matrix form it is easy to determine the corresponding angles [Paul 1981].

The quickest way to find a matrix, M, for any transformation is to investigate its effects on the standard basis.[21] Applying $q = [S, X, Y, Z]$ to element m in the standard basis yields the mth column of the matrix:

$$M = \begin{pmatrix} | & | & | \\ n & o & a \\ | & | & | \end{pmatrix} = \begin{pmatrix} 1-2Y^2-2Z^2 & 2XY-2SZ & 2XZ+2SY \\ 2XY+2SZ & 1-2X^2-2Z^2 & 2YZ-2SX \\ 2XZ-2SY & 2YZ+2SX & 1-2X^2-2Y^2 \end{pmatrix}. \quad (5.4)$$

To determine roll, pitch, and yaw, only seven of the matrix elements of equation 5.4 are required. If M_{ij} is the element occurring in the ith row and jth column of the matrix, then using the formulae given in [Paul 1981] yields:

$$\begin{aligned} roll &= \begin{cases} 0 & \text{if both } M_{11} \text{ and } M_{21} \text{ are 0} \\ atan(M_{21}, M_{11}) & \text{otherwise} \end{cases} \\ pitch &= atan(-M_{31}, M_{11}(\cos roll) + M_{21}(\sin roll))) \\ yaw &= atan(M_{13}(\sin roll) - M_{23}(\cos roll),\ M_{22}(\cos roll) - M_{12}(\sin roll)). \end{aligned}$$

Choosing the Knot Points

The formulae in the last two sections assumed t went continuously from 1 down to 0. When applying these to a machine, discrete points from within this interval will have to be used as knot points. Knot points need to be chosen in an application specific manner so that any error made on the path between the points is tolerable. In the Excalibur implementation, the user sets a parameter which indicates how often a new knot point is calculated. Taylor [Taylor 1986] presents a method for bounding the deviation from interpolated paths, by recursively halving the distance between knot points until a satisfactory deviation is obtained half-way between the points.

[21] In R^3 this is {(1, 0, 0), (0, 1, 0), (0, 0, 1)} = {**i**, **j**, **k**}.

The method reasonably assumes the mid-point deviation to be approximately the worst error over the segment.

Problems in Application to a Robot

Implementation of straight line motion on a manipulator brings to light many problems. These difficulties are primarily due to the geometry of the robot arm. Three such issues, whose solutions are often manipulator and task dependent, or non-existent, are now described.

Although a manipulator can reach both endpoints, the straight line joining them may contain points which cannot be attained (because it would cause the robot to move "through itself" or it requires a joint position beyond the limits of the robot). It is difficult to predict such conditions without calculating the line and checking that no points are out of reach. This is computationally expensive. Instead the thesis implementation starts the linear path, and skips the knot points which the Excalibur cannot reach. Obstacles also limit the positions a robot can safely attain. No obstacle avoidance has been implemented for the Excalibur.

Degenerate manipulator configurations and redundant configurations reaching the same position are another source of problems in any controlled motion. Cartesian interpolation breaks down under degeneracy [Paul 1981]. It is computationally expensive, if not impossible, to calculate all manipulator configurations which attain a knot position. Moreover which joint arrangement should be used for the smoothest motion? This is still an open research issue [Brady 1986].

Finally, small transitions in Cartesian position may cause huge changes in joint positions. This results in the manipulator moving irregularly. Even though the end effector travels linearly, delays in the motion may occur. The effects are difficult to predict [Paul 1981]. No solution to this problem exists.

5.3 Chapter Summary

This chapter began with previously unpublished kinematics for the Excalibur robot. They have been implemented and tested on the manipulator.[22] Implementation of the PMFs was discussed next by providing formulae for interpolating location and orientation. Linear interpolation in Cartesian three–space is well–known, hence concentration was on orientation for which quaternion methods were used. Rather than just presenting the formulae, an intuitive understanding of quaternions is encouraged by careful explanation showing how they relate to vectors, matrices, and roll, pitch, and yaw. Quaternion multiplication facilitates rotation, providing an alternative to the usual matrix techniques. That quaternions are superior was shown by comparison to matrices in standard tasks thematic of vector rotation. Quaternion interpolation has been successfully implemented on the Excalibur and an example was presented in the chapter.

[22] They have also been re–implemented numerous times through their use in the computer science robotics course and a robot simulation at the University of Calgary. A researcher at the Royal Military College in Kingston, Ontario, is also beginning an implementation of the solution presented in this thesis.

Chapter 6

Analysis and Future Directions

Recall the goal of this thesis — *to design and implement a system which learns robot tasks from examples.* ETAR's design is the topic of chapter 4 and thorough comparison to other acquisition systems is in chapter 3. Hence, this chapter concentrates on the implementation and parameters of the learning algorithm. ETAR is almost completely implemented, and the first section indicates the extent of this. The next section provides a performance indication by stating the execution times required in the block stacking example of the previous chapter. Since ETAR is an inductive learner, portraying it in the theoretical framework for induction is insightful. This is accomplished by developing a grammar for the learning system. Then the fundamental assumptions of each learning component are investigated and limitations are determined. In the end, the chapter is summarized, stressing the future directions which were proposed throughout.

6.1 Implementation Status

A test version of ETAR has been implemented on a Sun Microsystems network running the UNIX operating system. Most of the code is in Chez Scheme [Dybvig 1987], with the robot interface in C. Over 10,780 lines of code were written (about half of this number is documentation). This includes:

1. learning system. All components of the system are complete, except a history and workspace updating mechanism. When implemented, the updating mechanism will ensure that the frame information is consistent as objects are moved.[1] This is essential when the robot manipulates the same object more than once. The history mechanism will record important frames before they are updated, so that inner induction operates on temporally correct frames. Because these two mechanisms are missing, the program module which generates the loop and branch conditions has not yet been integrated with the rest of the system. In other words, ETAR proceeds nonstop from the input example to midway through inner induction (generating the PMF arguments). Then the loop/branch conditions are determined separately as the user updates the frames for the discrimination module. An execution module for running the acquired program must still be developed and implemented.

2. interface to robot. A serial interface routine was written in C to establish communication to the robot controller. Another interface contains a Scheme version of the robot commands, a parser for the feedback returned, functions to multiplex the serial input and output between files, and a teach interface which initializes the controller teach mechanism and guides the user in teaching. Coding the interface was very time consuming due to *many* hardware problems, and only successful once several errors were eliminated from the controller firmware.

3. forward and inverse kinematics. Kinematics are completely implemented and tested. The sample inverse solution from the manufacturer was proven incorrect for every joint, and only "works" for a small subset of possible manipulator configurations.

4. motion interpolation. The PMFs have all been implemented in anticipation of the execution module. A little known quaternion method is used to interpolate rotation in *rotate* and *transRot*.

[1] A vision system would eliminate the need for an updating mechanism.

component	time (ms)
single example to decimal joint angles[a]	1
single example to world coordinates, PMFs, and FOAs	82,400
intra–example processing on single example	150
inter–example matching of two examples	120
function induction merging two examples[b]	3,870,440
determining the loop termination condition	30

[a] A user required 101 seconds to perform the task and 1012 path interpolation points were collected.
[b] Assuming a single basis function, +

Figure 6.1: Execution times for ETAR on the stacking example

5. testing routines for each program module. All examples in chapters 2, 4, and 5 have been run by the implementation.

6.2 Testing

During the entire year and a half of this project, the available robot was unreliable and obtaining user teach sequences was next to impossible. Several sequences of the stacking task were obtained, and the conversion to PMFs was successful. ETAR's generalization components were executed on data simulated from these results, and the actual constructed task was shown in chapter 4. Generalization components were further tested on a variety of fabricated examples. However, the ultimate test for ETAR is people using it. Do people find ETAR easier than programming? How accurately does ETAR generate the task which the user intends? If the results from the stacking task are any indication, the answers are positive, but general user experiments on this task and others must be done for proper evaluation. Sorting objects and widget assembly are two tasks planned in the near future.

Figure 6.1 contains the execution times required by each component of the learning algorithm as the stacking task is acquired. All results are based on version 4.0 of SunOS running on a Sun4 computer. Efficiency was not a goal, and in the future, parts may be optimized.[2]

6.3 Theoretical Viewpoint

Theoretical evaluation was not part of the objectives of this research since the goal is to develop a system which helps users. But since ETAR is an inductive learner, it is important to place the system within the theoretical framework for inductive inference posited by [Gold 1967] (and summarized beautifully in [Angluin & Smith 1983]).

The criterion for success of inductive inference systems is *identification in the limit*. Suppose that a learner forms a conjecture, c_i, which is correct for all examples encountered so far. If the learner stops modifying the conjecture after a finite time, *i.e.*

$$c_t = c_{t+1} = c_{t+2} = \cdots,$$

then it has identified the concept in the limit. Using this criterion, it is well–known that learning in infinite domains is impossible with positive examples alone [Gold 1967]. Convergence is possible by several methods:

- presentation order. Proper presentation order makes learning in recursive languages possible from positive examples [Gold 1967]. However, ensuring a proper order is a difficult chore for a user and would violate one of the thesis objectives.

- negative examples. ETAR assumes that all examples are positive. A question for future work is whether user mistakes and/or intentional errors could benefit[3] learning. Processing negative examples will be difficult since isolating the erroneous component is complicated and cumulative errors may be present.

[2] In particular, as is mentioned later in this chapter, many researchers are working on optimal function induction algorithms which could replace ETAR's induction module.
[3] Note the term "benefit". Negative examples must never be necessary for ETAR though they may make learning easier.

The class of tasks learnable by ETAR can be formally represented by the grammar which uses the symbols:

::=	may be rewritten as	<x>	a non-terminal x
|	or	x^+	repetition of x one or more times
x°	repetition of x one or more times, without duplication of motions according to section 4.4.1	$::\stackrel{*}{=}$	may be rewritten as (robot / knowledge base dependent).

	<task>	::=	<statement>$^+$
	<simple task>	::=	<simple statement>$^+$
	<statement>	::=	<branch> | <loop> | <foaGroup>
	<simple statement>	::=	<foaGroup> | <simple branch>
	<loop>	::=	UNTIL <condition> <simple task>
	<branch>	::=	IF <condition> <task>
	<simple branch>	::=	IF <condition> <simple task>
†	<foaGroup>	::=	<imprecision> <precision>$^\circ$ | <precision>$^\circ$
	<condition>	::=	(slot, filler) ∈ object hierarchy
	<imprecision>	$::\stackrel{*}{=}$	moveto(<fn>, <fn>, <fn>, <fn>, <fn>, <fn>)
	<precision>	$::\stackrel{*}{=}$	rotate(<fn>, <fn>, <fn>) | translate(<fn>, <fn>, <fn>) | transRot(<fn>, <fn>, <fn>, <fn>, <fn>, <fn>) | grip(closed) | grip(open)
‡	<fn>	::=	<var> | <basis function>
	<basis function>	$::\stackrel{*}{=}$	plus(<fn>, <fn>) | minus(<fn>, <fn>) | multiply(<fn>, <fn>) | divide(<fn>, <fn>)
	<var>	::=	slot ∈ object hierarchy

†<foaGroup> yields motions which apply to the same FOA.
‡The nesting of functions <fn> is limited by a search complexity constraint. Currently, ETAR assumes that <fn> contains a maximum of 3 basis function calls.

Figure 6.2: Class of learnable tasks

- semantic model. Introducing a semantic model for the solution makes positive examples sufficient for learning [Angluin & Smith 1983, Wexler & Culicover 1980]. This is the category to which ETAR belongs. The model of the tasks ETAR can learn is shown in figure 6.2 as a context sensitive grammar. Every example is uniquely converted to a member of this grammar by the attention focussing mechanism and the PMF definitions. Matching and merging combine the examples in such a way that the general description remains in the grammar. Thus, the grammar restricts the possible task space, and the positive examples restrict the relevant members of the grammar, to arrive at the task description. These tasks are a sequence of PMFs, loops, and branches, where branches may be nested but loops are not. Further work must develop the acquisition of nested and counted loops. ETAR is adaptable to other robots, and the grammar clearly indicates those segments of ETAR which are application dependent.

Only controlled experiments will determine whether the model captures the user's intent. Future work must analyze how robot procedures should be structured and how they are best represented. Nevertheless, the control choices made in each part of the learning algorithm provide insight to the capabilities and limitations of ETAR, as the next section discusses.

6.4 Component–wise Investigation of ETAR

This section justifies the assumptions underlying the main components in ETAR's learning algorithm. Difficulties arising from the assumptions are stated and possible elaborations are indicated for future work.

6.4.1 PMFs

As the grammar indicates, the PMFs are robot specific and must be properly initialized. The PMF module is established from three considerations:

- feedback available from the robot. Excalibur feedback consists solely of its seven joint positions. When other feedback is available the robot's potential is increased. For example, if a robot has force sensors, some PMFs may be functions of force and not just position. Hence tasks which require a certain amount of force, *e.g.* chopping wood, may become feasible.

- capabilities of the robot. The robot motions provided at the controller level are a starting point for the PMFs. ETAR's *moveto* and *grip* directly initiate a controller command. The Excalibur controller contains an additional motion command, a relative joint level move, however it is rarely used and is subsumed by the *moveto*.

- capabilities required. Tasks often depend on precision motions not included with the robot controller, such as straight line traversal. Enhancing the PMFs with precision motions is necessary and the Excalibur implementation of ETAR contains *translate*, *rotate*, and *transRot*. These are the movements typically present in the programming languages of non–mobile robot arms and are a foundation for the tasks they perform. Other domains, *e.g.* ping–pong, require advanced PMFs which depend on other characteristics such as force and velocity. A downfall of including these PMFs is that their controller–level definition must be programmed in the PMF module of ETAR. However they are intended to be simple — the basis for higher level tasks acquired. Robot languages may already contain precision motions which can be transferred into ETAR.

A valuable project for the future is to try ETAR on other robots, particularly industrial robots such as the PUMA.

6.4.2 Focussing Mechanism

Centering attention on the necessary parts of a task is one of the most important contributions of ETAR. Any real world situation must be condensed to a small chunk of useful information before learning is feasible. Thorough investigation of appropriate focussing mechanisms warrants an entire thesis in itself and this section indicates some of these directions. Two mechanisms are part of ETAR:

- closeness to objects. Motion through empty space is ignored. Instead, ETAR concentrates on movement occurring in the vicinity of objects. This principle comes from observing people. Our daily tasks consist of displacing objects. As we move an object, our attention focuses on that object and those objects around it.[4] What the limbs do away from objects is irrelevant. Furthermore, the tasks done by people vary according to the characteristics of the objects being manipulated. It is therefore essential to exploit these features in any learning system. This is the job of the FOA in ETAR.

- guiding speed. People tend to slow down as more precision is required, especially when they are showing someone else how to perform. This speed–accuracy tradeoff has long been investigated in psychology, and ETAR exploits it to focus attention on the precise movements of a task. Thus, if the robot is quickly moving past close objects, they are ignored. "Quick" is user dependent since experienced users generally work faster than novices. A simple test at the beginning of any teaching session must be incorporated. This test will have the user move the arm through space and then pick up a known object. From this data a division between fast and slow (hence precision and imprecision) can be determined. A question which immediately emerges is: what if the task requires a certain amount of speed before it is successful, *e.g.* chopping wood? Here the axe is travelling fastest at the initial impact with the wood — an

[4] This does not preclude our attention focussing elsewhere as well.

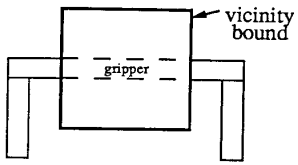


Figure 6.3: Vicinity distance too small

important point which must not be discarded. However, when such tasks are taught, the teacher often demonstrates the motion path very slowly, emphasizing the correct position of the objects. ETAR's strategy may still work, but further investigation is required.

Limitations of these focussing mechanisms are apparent. Consider hitting a ping–pong ball. This task depends on characteristics of the ball (*e.g.* location, weight, and material of the ball), but also on the desired final position. In another example, see figure 6.3, the robot may need to place a board on top of two posts which are far apart. ETAR's current FOA will not notice the posts. Further work on the FOA is required, and, in particular, the following must be investigated:

1. destination of an object. Characteristics of the area near an object's goal position may affect the initial movements of a task. This consideration is necessary in the exemplary ping–pong problem just proposed.

2. variable vicinity distance. When the robot is grasping an object, then the vicinity distance should expand around the grasped object and not just around the robot gripper. This would solve the board placing problem just posed. However, care must be taken to assure that the FOA does not become too large. Maulsby's touch relation [Maulsby 1988] is a specific example of this consideration.

3. speed of an object. Objects moving fast, especially those approaching the robot, may be in the FOA.

4. exceptions to expectation. If the robot has a pre–conception of the world in which it is operating and the sensory information does not correspond, then the differences are a focus of attention.

5. new arrivals to a scene. Similar to the previous point, as new objects arrive in a confined workspace, attention should turn to them.

6. instruction. Someone may point out an object which must always be in the FOA.

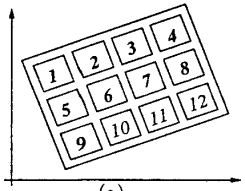
7. previous errors. Focus on objects noticed in earlier errors.

6.4.3 Loops

As indicated by the grammar, ETAR does not handle nested or counted loops. Instead, each loop continues until a learned condition occurs. This limitation is now discussed and directions to incorporate nested and counted loops into ETAR are investigated.

A common robot task with both nested and counted loops is shown in figure 6.4(a): pallet stacking. Part (b) of the figure shows a paraphrase of the desired task description. What does ETAR learn in this case? It spots the outermost loop, *i.e.* that each object is picked up and then placed somewhere on the pallet, as part (c) of figure 6.4 indicates. However, ETAR does not recognize the relationship between placing successive objects. This is not unexpected, since ETAR constructs relationships from the characteristics of the objects and not from previous positions as NODDY does. But positional parameters alone are not sufficient and NODDY cannot learn pallet stacking. More work is necessary to delimit when and how positional parameters can be extracted as part of the FOA in procedure learning.

Recall that loops are determined by repetition in key events and FOA groups. If ETAR were told that row 1, row 2, and row 3 are explicit parts to the pallet, then



(a)

```
LOOP UNTIL all objects stacked
    FOR row = 1 TO 3
        FOR column = 1 TO 4
            place object at (row, column) of pallet
            on top of previous object there
```

(b)

```
LOOP UNTIL all objects stacked
    place object on pallet
    (location undetermined)
```

(c)

```
LOOP
    place objects in row 1
LOOP
    place objects in row 2
LOOP
    place objects in row 3
LOOP
    place objects in row 1
    ⋮
```

(d)

```
LOOP UNTIL all objects stacked
    LOOP
        place objects in row 1
    LOOP
        place objects in row 2
    LOOP
        place objects in row 3
```

(e)

Figure 6.4: Pallet stacking

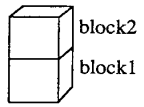


Figure 6.5: Problem with the FOA matching criterion: stack block1 onto block2

the task description learned differs to account for this. As shown in figure 6.4(d), ETAR will realize that a loop occurs as each row is placed. However, ETAR cannot determine a function for where an object is placed in the row (since this is a loop on the column positions), nor can it determine the loop termination condition (unless other information was supplied, such as "end of pallet" or "row full"). Once one level of the loop has been determined, ETAR could reapply it's loop search, this time searching for repetition in loop groups, to determine nested loops. Figure 6.4(e) shows this result. Clearly, more work is required in this area.

6.4.4 Matching

ETAR's matching criteria are simple, but strong. Are they powerful enough or possibly too restrictive? This question is now answered for the two rules which must be satisfied to match two FOA groups:

1. contain the same key events. This rule is necessary since motions which begin or stop the manipulation of objects must match to the same motion. *E.g.* if a robot grasps an object then the grasp must always match with a grasp. However, this rule alone is not sufficient because grasping an object to move it into a required position should not match with grasping an object to move it out of the way.

2. contain the same FOA type. Restricting FOA groups to match only when they contain the same explicit task parameters solves the aforementioned problem. Cases do arise when this constraint is too strong. Consider the extra stacking example in figure 6.5, where the order of the blocks must be reversed. Removing

symbol	meaning
F	number of basis functions
A	maximum arity of any basis function
n	complexity level
I	number of inputs

Number of argument expression forms with uninstantiated functions[a]

$$= \binom{An}{n} \frac{1}{(A-1)n+1}$$

Number of argument expressions instantiated with basis functions

$$= (\#\text{ forms}) \cdot (\#\text{ basis functions})^{(\#\text{ function placeholders})}$$

$$= \binom{An}{n} \frac{1}{(A-1)n+1} F^n$$

Number of uninstantiated arguments in an expression

$$= (\#\text{ arguments / function}) - (\#\text{ arguments filled as other functions})$$
$$= nA - (n-1)$$
$$= n(A-1)+1$$

Number of tests required to reject an argument expression

$$= I^{n(A-1)+1}$$

FUNCTION INDUCTION COMPLEXITY

Number of expressions tested

$$= \binom{An}{n} \frac{1}{(A-1)n+1} F^n I^{n(A-1)+1}$$

[a] Development of this formula is in [Knuth 1973]

Figure 6.6: Function induction complexity

block2 from block1 will not match with the clear loop generated from the previous stacking examples. Hence ETAR fails on this example. This problem is further magnified in teaching ETAR to build an arch. If the user constructs the right pillar first in one example and the left pillar first in the other, then ETAR will not match the two examples. Again learning fails. ETAR does not recognize symmetries and future work must investigate this problem. Incorporation with planners may be helpful, especially since the pre– and post–conditions are explicitly present in every task which ETAR learns.

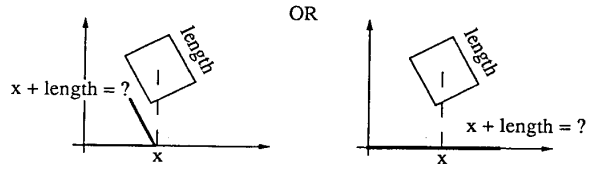


Figure 6.7: A meaningless test

6.4.5 Function Induction

Figure 6.1 showed that the time required for the function induction portion of ETAR greatly exceeds any other step in the algorithm. This is because the implemented version is only simple "generate and test". Many other researchers have developed algorithms which could replace this module, and reduce the time immensely, making the induction feasible (refer to [Falkenhain & Michalski 1986], [Kokar 1986], [MacDonald 1989], and [Phan 1989]). Their strategies include

1. accounting for algebraic properties such as symmetry and distributivity to constrain the space of expression forms. Canonical representations are often used [Falkenhain & Michalski 1986, MacDonald 1989].

2. type constraints. ETAR distinguishes numeric and symbolic types but that is the extent of its type checking. As summarized in chapter 3, COPER [Kokar 1986] constrains its search by dimensional units. Dimensional analysis would be advantageous to ETAR, since it avoids many meaningless expressions, such as $(height \cdot length)\text{cm}^2$ for a robot position in cm. Furthermore, ETAR generates other meaningless expressions such as $x + length$ shown in figure 6.7. The problem here is that x and *length* are really *directional*, and ETAR treats all numeric quantities as scalar unless they are explicitly expressed as a vector. One solution is to force all numbers to be vectors; another is to explicate the fact through a type tag field, as NODDY's *x_pos* and *y_pos* does.

3. limit the input. ETAR does this by its FOA mechanism, and as already mentioned, further work in this area is required.

An upper bound for ETAR's current function induction complexity is in figure 6.6 [MacDonald 1989]. This bound is exact when the arity of all the functions is the same and the required argument expression is not found at the level. Adding the aforementioned constraints will greatly improve this bound.

6.5 In the Future

This chapter highlighted the accomplishments of this thesis work — from the implementation of low level kinematics through to a complex robot task acquisition system. Execution statistics were given for each learning component as the system was tested on the familiar block stacking example. Then, it was shown that ETAR's components fashion a model for the tasks which can be learned. This model was represented as a context sensitive grammar, in which positive examples guide the search to the correct task description. Finally, each of the major individual components was analyzed to justify its assumptions and determine limitations. As limitations were noted, future directions were suggested. In summary these are:

1. experiments and user evaluation of the system
2. nested and counted loops
3. incorporation with planning, especially to handle symmetries
4. implementation on other (industrial) robots
5. further tasks, *e.g.* widget construction, building arches, sorting objects
6. FOA elaboration
7. incorporation of a perception system.

Chapter 7

Conclusions

This thesis is an important advance in useable, and hence more useful, robots. As a result of developing and implementing a task acquisition system (ETAR), the thesis demonstrates the feasibility of constructing new programs from examples in which a user directly leads the robot. This opens an alternate avenue to robot task specification. Rather than an elite group of experts programming the tasks, any person who is able to do a task with the robot can delegate it to the robot. Tasks which ETAR acquires are not only repeated, fixed sequences, but also standard assembly tasks such as widget assembly and block stacking. ETAR has been tested on one complete example and results are given throughout the thesis. Further refinement, more tasks, and experimental testing on a user population are challenges for the future.

ETAR is a prototypical learning by example system. It develops two novel ideas:

1. The numeric feedback supplied by the robot at lead time is partitioned into the robot's primitive motions. The result is that each example is modelled as a sequence of symbolic forms.

2. The entire generalization algorithm is rooted in the focus of attention — the objects important in a task. Repeated patterns in the focus of attention and in the key symbolic forms yield loops. The focus of attention and key events also control the matching of symbolic forms between examples. After matching, function induction fully merges the examples by constructing functions from characteristics of the manipulated objects. Thus, ETAR introduces structure — sequencing, loops, conditionals, and variables — to the tasks acquired.

Additionally, ETAR integrates four necessary characteristics of a robot learning system:

1. ETAR has been implemented on a real robot — the Excalibur. Although the manipulator was unreliable during the entire course of this work, limited tests for one example were obtained. These showed that learning by example is possible on real robots.

2. Examples are provided to ETAR through a direct lead mechanism which is widely available in robot systems. Thus, the user has an easily accessible visual interface to the task being constructed. The user does not encounter the difficult chore of designing symbolic traces of the task because ETAR builds them from the numeric feedback supplied by the robot.

3. ETAR requires minimal background knowledge. The workspace is represented as a frame system which explicates objects' characteristics (*e.g.* position, orientation). This information is normally present in computer–aided design, and furthermore, it would be replaced by perception systems if they were available.

4. ETAR is an inductive learner and acquires new tasks. Functional descriptions about the tasks, as in deductive learners, are not required.

Ideally, all that a robot learner should require of users is the ability to do the task. ETAR is only a beginning in this direction. The FOA mechanism was intended to meet this objective, and the user is not responsible for presentation order, keyboard entry of partial traces, or pointing out the relevant attributes. However, this thesis has only proposed a preliminary FOA which must be evaluated to see how it captures the intention of human tasks, how it limits the tasks which may be learned, and how it may be expanded to account for a greater variety of tasks.

As a prerequisite to the ETAR system, this thesis also contributes to low level robotics. Since ETAR is implemented on a real robot, it begins with joint data which must be converted into world coordinates, and on robot program execution, back into joint data. The thesis provides complete kinematic detail for the Excalibur robot. These results have never been previously published. Furthermore, it is doubt-

ful whether the inverse kinematics have ever been solved, since the manufacturer's sample code contains many errors. The solution developed in the thesis has been completely implemented, tested, and used by other people. When a learned task must be executed, the primitive motion functions are translated into robot joint control. Full details of this translation are given, with emphasis on straight line motion, in particular, smoothly changing orientation. A recently revived quaternion method is used and the unique introduction provides intuitive insight into quaternion rotation.

ETAR is incorporated into the foundations of learning by example by a careful survey of the field. ETAR learns open, active knowledge, *i.e.* how to *do* something in a *real* environment with entities different from the manipulator. This is essential in all robot learning systems, but has been neglected by most learners which only opt to classify internal symbols.

ETAR is not the ultimate learning system, and limitations are clearly presented in the thesis. However, it sets a precedent for a new generation of robot use — demonstrating examples instead of programming. All robots of the future should have this ability, which should be extended so that they will be adaptable to our dynamic world.

Appendix A

Notes on Formulae Used in Operations Counts

This appendix provides extra reference to the formulae used in calculating operation complexity for figure 5.19.

Rotating a Vector

Quaternion: Using the formula given in figure 5.16.
Matrices: Pre–multiplying a three–vector by a 3×3 matrix.

Composition of Rotations

Quaternion: Using the formula from figure 5.14.
Matrices: The first two columns obtained by matrix multiplication and the last column as the cross product of the first two.

Setting up the rotation from an angle and a unit axis

Quaternion: Specification of q as in figure 5.16.
Matrices: Formula on page 28 of [Paul 1981].

Extracting angle and axis from rotation

Quaternion: Inverting specification of q in figure 5.16.
Matrices: Method on page 19 of [Funda 1988].

Interpolation — finding the next rotational knot point

Quaternion: The calculation assumes that the endpoint quaternions have already been scaled by $\sin\theta$. An extra 6 M and 1 Trig are necessary for this once at the beginning of interpolation.
Matrices: It is unclear what the most efficient formulation of matrix–based orientation interpolation is. If the two endpoints are expressed as the matrices M_1 and M_2 then $M_{int} = M_1^T M_2$ is the matrix which changes M_1 into M_2. As t goes from zero to one the interpolated orientation is

$$M_1[f(t)M_{int}]$$

where $f(t)$ is a continuous function with $f(1) = 1$ and $f(0) = 0$. This formula is used for the operations counts, which may be higher depending on $f(t)$.

Bibliography

[Adams 1976] J. A. Adams. Issues for a closed–loop theory of motor learning. In G. E. Stelmach, editor, *Motor Control: Issues and Trends*, chapter 4, pages 87–107. Academic Press, Inc., 1976.

[Aken & Brussel 1988] L. Van Aken and H. Van Brussel. Robot programming languages: the statement of a problem. *Robotica*, 6:141–148, 1988.

[Amarel 1986] Saul Amarel. Program synthesis as a theory formation task. In R. S. Michalski, J. G. Carbonell, and T.M. Nitchell, editors, *Machine Learning*, volume 2, chapter 18, pages 499–569. Morgan Kaufmann Inc., Los Altos, CA, 1986.

[Ambler *et al* 1982] A. P. Ambler, R. J. Popplestone, and K.G. Kempf. An experiment in the offline programming of robots. In *Proc. of 6th Intl. Conf. on Industrial Robot Technology*, Paris, France, June 1982. IEEE.

[Andersson 1987] R. L. Andersson. *Real time expert system to control a robot.* PhD thesis, Department of Computer and Information Sciences, University of Pennsylvania, 1987.

[Andreae 1984] P. M. Andreae. *Justified generalization: acquiring procedures from examples.* PhD thesis, MIT, Department of Engineering and Computer Science, 1984.

[Andreae 1988] J. H. Andreae. Programming intelligent robots. *Canadian Artificial Intelligence*, pages 13–14, January 1988.

[Angluin & Smith 1983] D. Angluin and C. H. Smith. Inductive inference: theory and methods. *Computing Surveys*, 15(3):237–269, September 1983.

[Anton 1981] H. Anton. *Elementary linear algebra.* John Wiley and Sons, Toronto, 1981.

[Bartle 1976] R. G. Bartle. *The elements of real analysis.* John Wiley and Sons, Toronto, 1976.

[Biermann 1984] A. W. Biermann. Dealing with search. In A.W. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic Program Construction Techniques*, chapter 17, pages 375–392. Macmillan Publishing Company, 1984.

[Biermann *et al* 1984] A. W. Biermann, G. Guiho, and Y. Kodratoff. An overview of automatic program construction techniques. In A.W. Biermann, G. Guiho, and Y. Kodratoff, editors, *Automatic Program Construction Techniques*, chapter 1, pages 3–30. Macmillan Publishing Company, 1984.

[Bonner & Shin 1982] S. Bonner and K. G. Shin. A comparative study of robot languages. *IEEE Computer*, pages 82–96, December 1982.

[Brady 1986] M. Brady. Trajectory planning. In M. Brady, J.M. Hollerbach, T.L. Johnson, and T. Lozano-Perez, editors, *Robot Motion: Planning and Control.* The MIT Press, 1986.

[Capek 1923] K. Capek. *Rossum's universal robots.* Doubleday, Page, and Co., Garden City, New York, 1923.

[Carbonell *et al* 1986] J. G. Carbonell, R. S. Michalski, and T. M. Mitchell. An overview of machine learning. In R. S. Michalski, J. G. Carbonell, and T. M. Nitchell, editors, *Machine Learning*, volume 1, chapter 1, pages 3–23. Morgan Kaufmann Inc., Los Altos, CA, 1986.

[Cendrowska 1988] J. Cendrowska. Prism: an algorithm for inducing modular rules. *International Journal of Man–Machine Studies*, 27:349–370, 1988.

[Cohen & Feigenbaum 1982] P. R. Cohen and E. A. Feigenbaum, editors. *The handbook of artificial intelligence*, volume III. William Kaufmann, Los Altos, California, 1982.

[DeJong & Mooney 1986] G. DeJong and R. Mooney. Explanation-based learning: an alternative view. *Machine Learning*, 1(2):145–176, 1986.

[Dietterich 1983] T. G. Dietterich. A comparative review of selected methods for learning from examples. In R. S. Michalski, J. G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, chapter 3, pages 41–81. Tioga, Palo Alto, CA, 1983.

[Dietterich & Michalski 1986] T. G. Dietterich and R. S. Michalski. Learning to predict sequences. In R. S. Michalski, J. G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 2, chapter 4, pages 63–106. Morgan Kaufmann, Los Altos, CA, 1986.

[Dybvig 1987] R. K. Dybvig. *The* SCHEME *programming language.* Prentice–Hall, 1987.

[Excalibur 1986] Excalibur user's manual. RSI Robotic Systems International, 1986.

[Falkenhain & Michalski 1986] B. C. Falkenhainer and R. S. Michalski. Integrating quantitative and qualitative discovery: the ABACUS system. *Machine Learning*, 1:367–401, 1986.

[Fikes 1972] R. E. Fikes. Monitored execution of robot plans produced by STRIPS. *Information Processing*, pages 189–194, 1972.

[Fikes & Nilsson 1971] R. E. Fikes and N. J. Nilsson. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.

[Funda 1988] J. Funda. Quaternions and homogeneous transforms in robotics. Master's thesis, Department of Computer and Information Science, University of Pennsylvania, Philadelphia, 1988.

[Gini & Gini 1985] Giuseppina Gini and Maria Gini. Robot languages in the eighties. In K. Rathmill, P. MacConaill, S. O'Leary, and J. Browne, editors, *Robot Technology and Applications*, pages 126–138. Springer–Verlag, 1985.

[Gold 1967] E. M. Gold. Language identification in the limit. *Information and Control*, 10:447–474, 1967.

[Grossman 1987] S. I. Grossman. *Elementary linear algebra.* Wadsworth Publishing Company, 1987.

[Gustafson 1986] B. K. Gustafson. Development of localized planner for AI–based robot task planning system. Master's thesis, University of Illinois at Urbana–Champaign, Department of Mechanical Engineering, 1986.

[Hamilton 1969] Sir W. R. Hamilton. *Elements of quaternions.* Chelsea Publishing Company, 3rd edition, 1969. original manuscript 1865.

[Heise & MacDonald 1988] R. Heise and B. A. MacDonald. Kinematics of an elbow manipulator with forearm rotation: The excalibur. Research Report 88/334/36, Department of Computer Science, University of Calgary, October 1988.

[Heise & MacDonald 1989a] R. Heise and B. A. MacDonald. Quaternions and motion interpolation: a tutorial. In *Computer Graphics International*, Leeds, England, 1989. Springer-Verlag. [Also Research Report 88/329/41, Department of Computer Science, University of Calgary].

[Heise & MacDonald 1989b] R. Heise and B. A. MacDonald. Robot program construction from examples. In *Proceedings of AI / CS '89*, Dublin, Ireland, September 1989.

[Heise & MacDonald 1989c] R. Heise and B. A. MacDonald. Robots acquiring tasks from examples. In *Proceedings of the 2nd International Symposium on Artificial Intelligence*, Monterrey, Mexico, October 1989.

[Herstein 1975] I. N. Herstein. *Topics in algebra.* Wiley and Sons, Toronto, 1975.

[Holding 1981] D. H. Holding. Skills research. In D. H. Holding, editor, *Human Skills*, chapter 1, pages 1–13. John Wiley and Sons, 1981.

[Iba & Langley 1987] W. Iba and P. Langley. A computation theory of motor learning. *Computational Intelligence*, 3:338–350, 1987.

[Kadie 1988] C. M. Kadie. DIFFY-S: learning robot operator schemata from examples. In *Proceedings of the Fifth International Conference on Machine Learning*, pages 430–436, Ann Arbor, Michigan, 1988. Morgan Kaufmann Publishers, Inc.

[Kelland & Tait 1904] P. Kelland and P. G. Tait. *Introduction to quaternions.* The MacMillan Company, New York, 1904.

[Knox 1984] C. S. Knox. *Engineering documentation for CAD/CAM applications.* Marcel Dekker, Inc., New York, 1984.

[Knuth 1973] D. E. Knuth. *The art of computer programming*, volume I: Fundamental algorithms. Addison–Wesley, 1973.

[Kokar 1986] M. M. Kokar. Determining arguments of invariant functional descriptions. *Machine Learning*, 1(4):403–422, 1986.

[Langley *et al* 1986] P. Langley, J. M. Zytkow, H. A. Simon, and G. L. Bradshaw. The search for regularity: four aspects of scientific discovery. In R. S. Michalski, J. G. Carbonell, and T.M. Mitchell, editors, *Machine Learning*, volume 2, chapter 16, pages 425–469. Morgan Kaufmann, Los Altos, CA, 1986.

[Lozano-Perez 1983] T. Lozano-Perez. Robot programming. *Proceedings of the IEEE*, 71(7):821–841, July 1983.

[MacDonald 1989] B. A. MacDonald. Inducing expressions from examples. Research report, Department of Computer Science, University of Calgary, Calgary, Alberta, 1989. [In preparation].

[Maulsby 1988] D. L. Maulsby. Inducing procedures interactively. Master's thesis, University of Calgary, Calgary, Alberta, Canada, December 1988.

[Medland & Burnett 1986] A. J. Medland and P. Burnett. *CAD/CAM in practice.* Kogan Page, London, 1986.

[Michalski & Chilausky 1980] R. S. Michalski and R. l. Chilausky. Learning by being told and learning from examples: an experimental comparison of two methods of knowledge in the context of developing an expert system for soybean disease diagnosis. *Int. Journal Policy Analysis and Information Systems*, 4(4), 1980.

[Michalski *et al* 1986] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, editors. *Machine learning: an artificial intelligence approach*, volume II. Morgan Kaufmann, 1986.

[Minsky 1977] M. Minsky. Frame–system theory. In P. N. Johnson-Laird and P. C. Wason, editors, *Thinking.* Cambridge University Press, 1977.

[Mitchell 1977] T. M. Mitchell. Version spaces: a candidate elimination approach to rule learning. In *Proceedings of the Fifth IJCAI*, 1977.

[Mitchell 1982] T. M. Mitchell. Generalization as search. *Artificial Intelligence*, 18:203–206, 1982.

[Mitchell *et al* 1981] T. M. Mitchell, P. E. Utgoff, B. Nudel, and R. Banerji. Learning problem–solving heuristics through practice. In *Proceedings of the 7th IJCAI*, pages 127–134, 1981.

[Mitchell *et al* 1986a] T. M. Mitchell, J. G. Carbonell, and R. S. Michalski, editors. *Machine learning: a guide to current research.* Kluwer Academic Publishers, 1986.

[Mitchell *et al* 1986b] T. M. Mitchell, R. M. Keller, and S. T. Kedar-Cabelli. Explanation-based generalization: a unifying view. *Machine Learning*, 1(1):47–80, 1986.

[Moravec 1988] Hans Moravec. *Mind children: the future of robot and human intelligence.* Harvard University Press, Cambridge, Massachusetts, 1988.

[Newell 1981] K. M. Newell. Skill learning. In D. H. Holding, editor, *Human Skills*, chapter 9, pages 203–226. John Wiley and Sons Ltd., 1981.

[Parnas 1987] D. L. Parnas. Why engineers should not use artificial intelligence. In *CIPS Proceedings*, pages 39–42, Edmonton, 1987.

[Paul 1981] R. P. Paul. *Robot manipulators: mathematics, programming, and control.* The MIT Press, 1981.

[Pauli & MacDonald 1988] D. Pauli and B. A. MacDonald. Inducing functions in robot domains. Research Report 88/296/08, Department of Computer Science, University of Calgary, Calgary, Alberta, 1988.

[Phan 1989] T. H. Phan. The equal–value search: accelerating search in function induction. Master's thesis, The University of Calgary, Calgary, Alberta, April 1989.

[Poppleston *et al* 1980] R. J. Popplestone, A. P. Ambler, and I. M. Bellos. An interpreter for a language for describing assemblies. *Artificial Intelligence*, 14:79–107, 1980.

[Quinlan 1983] J. R. Quinlan. Learning efficient classification procedures and their application to chess end games. In R. S. Michalski, J. G. Carbonell, and T.M. Mitchell, editors, *Machine Learning: An Artificial Intelligence Approach*, volume 1, chapter 15, pages 463–482. Tioga, Palo Alto, CA, 1983.

[Quinlan 1986] J. R. Quinlan. Induction of decision trees. *Machine Learning*, 1(1):81–106, 1986.

[Rich 1983] E. Rich. *Artificial intelligence.* McGraw–Hill, 1983.

[Rock 1988] S. T. Rock. Intelligent robot programming: you can't get there from here — a viewpoint. *Robotica*, 6:333–338, 1988.

[Schmidt 1976] R. A. Schmidt. The schema as a solution to some persistent problems in motor learning theory. In G. E. Stelmach, editor, *Motor Control: Issues and Trends*, chapter 2, pages 41–65. Academic Press, Inc., 1976.

[Schmidt 1982] R. A. Schmidt. *Motor control and learning — a behavioral emphasis.* Human Kinetics Publishers, Champaign, Illinois, 1982.

[Segre 1988] A. M. Segre. *Machine learning of robot assembly plans.* Kluwer Academic Publishers, 1988. [Also Ph.D. thesis, Department of Electrical and Computer Engineering, University of Illinois at Urbana–Champaign, 1988].

[Shoemake 1985] K. Shoemake. Animating rotation with quaternion curves. *Computer Graphics*, 19(3), 1985.

[Takase *et al* 1981] K. Takase, R. P. Paul, and E. J. Berg. A structured approach to robot programming and teaching. *IEEE Transactions on Systems, Man and Cybernetics*, SMC–11(4):274–289, April 1981.

[Taylor 1986] R. H. Taylor. Planning and execution of straight–line manipulator trajectories. In M. Brady, J. M. Hollerbach, T. L. Johnson, and T. Lozano-Perez, editors, *Robot Motion: Planning and Control.* The MIT Press, 1986.

[Teicholz 1985] E. Teicholz, editor. *CAD/CAM handbook.* McGraw–Hill, 1985.

[TIScheme 1987] TI scheme language reference manual. Texas Instruments Incorporated, 1987.

[VanLehn 1983] K. VanLehn. Felicity conditions for human skill acquisition: validating an AI–based theory. Interim Report P8300048, XEROX Palo Alto Research Centers, 1983.

[Weisbin *et al* 1989] C. R. Weisbin, G. de Saussure, J. R. Einstein, F. G. Pin, and E. Heer. Autonomous mobile robot navigation and learning. *Computer*, 22(6):29–35, June 1989.

[Wexler & Culicover 1980] K. Wexler and P. Culicover. *Formal principles of language acquisition.* MIT Press, Cambridge, Mass., 1980.

[Witten & MacDonald 1988] I. H. Witten and B. A. MacDonald. Using concept learning for knowledge acquisition. *Int. J. Man-Machine Studies*, 29(2):171–196, August 1988. [Also presented at AAAI Knowledge Acquisition for Knowledge-Based Systems Workshop, Banff, Canada].