MASSACHUSETTS INSTITUTE OF TECHNOLOGY

A.I. LABORATORY

Artificial Intelligence
Memo no. 301

January 1974

A MECHANICAL ARM CONTROL SYSTEM

Richard C. Waters

ABSTRACT

This paper describes a proposed mechanical arm control system and some of the lines of thought which led to this design. In particular, the paper discusses the basic system required in order for the arm to control its environment, and deal with error situations which arise. In addition the paper discusses the system needed to control the motion of the arm using the computed torque drive method, and force feedback.

Work reported herein was conducted at the Artificial Intelligence laboratory, a Massachusetts Institute of Technology research program supported in part by the Advanced Research Projects Agency of the Department of Defense and monitored by the Office of Naval Research under Contract Number N00014-70-A-0362-0005.

TABLE OF CONTENTS

I. INTRODUCTION

The M.I.T. A.I. laboratory is in the process of producing a small robot manipulation system, the 'mini-robot'. The overall structure of this system is dominated by a division into three parts:

a) A high power remote computer which 'thinks'. That is, the computer decides on a course of action based on the information available to it.

b) A small local computer which controls a vision system. This system is the basic sensor of the mini-robot, and is used to gather information about the environment. This information is then sent to the high level system.

c) A small local computer which controls a mechanical arm. The arm is the mini-robot's output device, and performs 'actions', effecting changes in the environment. Though it can also be used as a sense organ (for forces), its abilities in this regard are very primitive.

It should be stressed that the high power computer probably will be physically removed from the local computer(s). Further it is one of the basic design criteria that the bandwidth needed to communicate between the two computer systems be as small as possible.

This paper is concerned with the local software associated with the mechanical arm. This being the case, the high level and vision systems are only mentioned in passing. However, the reader should keep their existance in mind.

Since the mechanical arm is basically just an output device, its control system minimally could be very simple and devoid of

'intelligence'. All of the intelligence could be isolated in the high level system. On the other hand, a lot of information and intelligence dealing with 'actions' could be placed in the arm control system so that the high level system could operate at a more symbolic level. This is the approach taken here. It is intended that the requests for action by the high level system will be concise orders for complete actions like "pick up the ball", or "throw the ball".

The paper is organized as follows: Section II first makes some general comments about control which motivate the design decisions made in the rest of the paper. It then tries to indicate what the arm controller is capable of doing on its own.

Section III specifically states the behaviour it was decided that the arm controller should exhibit. It also attempts to explain why particular design decisions were made. Finally, scenerios of four typical arm actions are given in order to make the discussion more concrete. Electronic circuit board construction has been chosen as a prototypical task for the mini-robot, and all of the examples in this paper are taken from this domain.

Sections IV-VI describe a proposed mechanical arm control system which exhibits the required behaviour. This is the meat of the paper in that it contains the specific details of the proposed system. However the reasons behind the proposed system are vitally important as well.

II. SOME IDEAS ON CONTROL

II.1 'REGULATE' VS 'CONTROL'

In this paper, 'regulate' and 'control' are used in a special sense to express two ends of a spectrum of the degree of 'goodness' of control. If A regulates B then the control A has over B is so complete absolute and immediate that it is not thought of as being fallible or as being a process, though it probably is. If A regulates B then B is said to be a 'direct effect' of A.

For example, a person 'regulates' the position of the thermostat in a room. In contrast, by means of this thermostat, he 'controls' the temperature of the room. This is true even though the processes which allow a person to regulate the position of the thermostat are infinitely more complex than the process which allows the thermostat to control the temperature. It is not the complexity of the control process, but its effectiveness that is important here.

As a second example, the thermostat mechanism regulates the position of a switch. This switch in turn regulates the on/off state of the furnace. Finally, the on/off state of the furnace affects the temperature (with a large time delay). Through this sequence of effects, the thermostat is able to loosely control the temperature. This illustrates the interesting phenomenon that as you look into a control mechanism, the direct effects become more primitive and farther removed from the goals of the mechanism.

II.2 'OBSERVABLES'

For A to control B, A must regulate something which, through a chain of effects, eventually affects B. In addition, unless A is the only factor affecting B, A must be able to observe B, or at least something related to B.

Here again, the thermostat is a good example. It was designed with the realization that many factors affect the temperature, and therefore was designed to measure the temperature. However, in other parts of the control loop the designers were not so careful. They assumed that the on/off switch has complete control over the furnace and therefore the thermostat was not designed to monitor the furnace. Thus when the burner blows out, or the oil runs out, a thermostat looses control of the temperature.

On a higher level the thermostat is only partially effective, because what a person really wants controlled is his comfort, not the temperature. His comfort is only tenuously related to the temperature near him, which, in turn, is only loosely related to the temperature near the thermostat. The thermostat is only partially effective because it cannot observe what a person really wants it to control. It has to settle for something only very indirectly related.

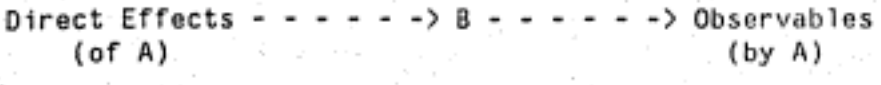

Fig. 1 Schematic of a control situation.

The three key factors of a control situation are shown in figure 1. The more tenuous the chain of effects which link them, the weaker the control.

II.3 THE MECHANICAL ARM

Let us consider the mechanical arm control problem in the light of the above ideas. The first thing to notice is that there are really two control problems:

a) How is the arm going to control things in its environment? For example how will the arm assemble a circuit board?

b) How will the arm controller control the arm? For example how will it make the arm move from point X to point Y?

II.3.1 ARM AS CONTROLLER

Looking at the first problem, it is not usually thought of as a 'control' problem. This is because the connections in Fig. 1 are so tenuous. Firstly, the direct effects of the arm are forces applied to objects in its environment and the motion of the arm through the environment (possibly moving something). These effects have almost no bearing on the state of completion of a circuit board. Only through a very complex procedure can they cause the assembly of a circuit board.

Secondly, the arm can only observe forces applied to it, such as contact with other objects, and its own position. This does not enable the arm to draw hardly any conclusions about its environment. In the prototype problem of circuit board construction, the general inability of the arm to observe its environment is overshadowed by the great difficulty in observing the state of completion of a circuit board by any means.

What does "observe the state of completion of a circuit board" mean? Clearly vision is required to find out where all of the components and other bits and pieces on the work bench are. However this list of where each thing is is not a measure of the state of completion. There must also be a statement of what a completed circuit board is like. From these two things, a list of all the differences between the observed circuit board, and a completed one can be created. This difference list seems to describe the situation, but it is still somewhat unsatisfactory. Not only is it cumbersome, it does not answer the question "what should be done next". The order in which the components are assembled is often not obvious but important. People constructing circuit boards usually follow an explicit procedure which says what to do when. They gauge the state of completion by how far they have gotten in the procedure.

We see that the closest thing there is to a measure of state of completion is the point in the construction procedure the builder has reached. Unfortunately this is not a very good measure. It allows the builder to talk about and deal with the states of completion that the

procedure causes the circuit board to pass through. However, if the normal course of events is disturbed and the circuit board is not in one of the expected states then this measure cannot even describe the situation let alone indicate what should be done about it.

The great strength of the thermostat on the other hand is that it is prepared to deal (perhaps crudely) with any possible value of the temperature.

In order to be 'intelligent', the mini-robot as a whole must have the ability to deal with almost any eventuality. It must have a description good enough to describe all situations, and be able to find a way out of any trouble. However, as discussed in section II below, the arm controller probably should not have this kind of power. To the extent that it is only executing plans of action, it cannot be said to be in control of its environment, because it is helpless except in very simple error situations.

Looking at the individual steps in a circuit board construction procedure, things look much more like a control problem. Take a relatively complex step like "move the 4K resistor now at point A to point X". Here, due to the greater inherent simplicity, the arm can observe the relevant state of the world (i.e. arm position). No matter what happens, it can simply continue to reduce the distance between the hand and the resistor (if it is not holding the resistor) or between the hand and point X (if it is holding the resistor). It can deal with everything that can happen, because not very much can happen. Note that if the arm drops the resistor, insurmountable problems arise. Without

vision the arm cannot find the resistor in reasonable time, or be sure it has found it and not the 10K resistor.

It is still executing a procedure ((MOVE_TO A) (PICK_UP) (MOVE_TO X) (RELEASE)), and the position in this procedure is being used as a measure. However, there is a key difference. The procedure is simple, but more importantly, it is such that any state of the system can be described as a point in the (parameterized) procedure. Thus the controller always knows what to do (i.e. finish the procedure).

This paper is entitled "A MECHANICAL ARM CONTROL SYSTEM". This is really rather ambiguous. Its meaning depends on whether the arm is seen as controller or controlled. The discussion above centers on its use as a controller, which is the area of greatest interest. (It should be noted that the rest of this paper concentrates on control corresponding to single steps in a procedure, rather than on the high level control of the environment, which is accomplished through the production of appropriate procedures from the single step building blocks). Having said this, it must still be recognized that the arm cannot be a controller if it is not controlled.

II.3.2 CONTROLLING THE ARM

A mechanism which is trying to control the arm can regulate the torque delivered by the motors at the arm's joints. What the mechanism is trying to control is the arm's position, velocity, and the force it exerts on external objects. Fortunately the control mechanism can more or less directly observe the position and force, which makes one of the

links in Fig. 1 very short. Unfortunately the motor torques, though functionally related to the position etc., have a very complex relation to them. This makes it very hard (in the sense of mathematically complex, not logically difficult) for a control mechanism to calculate how to change the torques in order to counteract an observed error.

It is interesting to note that there are many ways the control problems could be simplified by improving the direct effects or observables available to the controller. For applying a force with the unmoving arm, the situation is already optimal. The arm controller observes the force, and directly effects the torques. For motions of the arm, the analogous optimal situation would be for the controller to observe both the position and the velocity and directly effect the acceleration (instead of the torques). Note that if this was the only mode of control then the control mechanism would be unable to control the force exerted by the arm on external objects. Ideally the controller should be able to switch between these two methods of control. The situation would be almost as good if the controller could observe the acceleration, because instead of using huge equations, it could take advantage of the fact that the torque at a joint is monotonically related to the acceleration at that joint. However it might still want to be able to use the predictive power of the equations.

III. THE BASIC CAPABILITIES OF THE ARM CONTROLLER

III.1 HOW 'INTELLIGENT' SHOULD THE ARM CONTROL SYSTEM BE?

The first question to be asked about the arm control system is how 'intelligent' should it be? Intelligent is a hard word to define. In this context the question can be rephrased as: how complex are the actions which the system can reliably control, despite the possibility of errors and unexpected events.

Thus there are two major kinds of 'intelligence' the system should have. First the system should have knowledge of many complex actions which can be initiated with concise commands from the higher level systems. This knowledge can best be stored as procedures.

The other type of 'intelligence' deals with extraordinary situations. What should the system do when something goes wrong? More basically, what is it possible for the system to do based on the information it receives and its abilities. The arm controller cannot deal with situations it cannot effect or cannot detect. Thus looking at the direct effects, and the observables available to the local system indicates how much 'intelligence' is possible.

Through its direct effects, the arm can move an object to an arbitrary point and orientation in its working volume, and then apply a force. This should allow the arm to perform tasks such as circuit board construction.

The observables are a different matter. The arm's sense of touch does not give it much information. It must look to the rest of the

mini-robot system for most of the key information it needs.

III.2 WHAT INFORMATION LINKS SHOULD THERE BE?

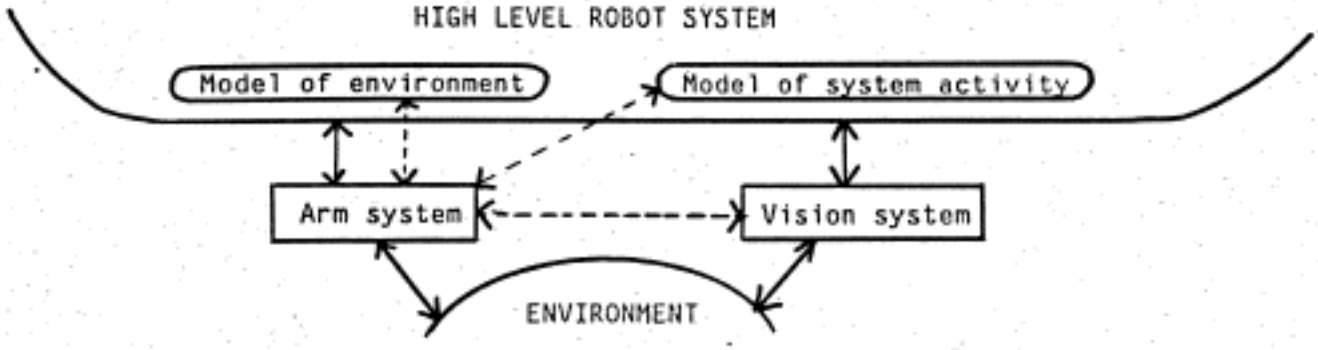

Fig. 2 Diagram of the communication links in the Mini-Robot system. Solid lines indicate definite communication links. Dotted lines indicate possible additional links.

Fig. 2 shows several possible information links the arm system could have with other components of the mini-robot system. Which ones should be implemented?

The answer to this question is dominated by the fact that since the arm controller may be physically removed from the rest of the system, high speed communication may not be possible. Further, the simplification gained by reducing the interaction with the rest of the system is certainly desirable, if the degradation in the arm's abilities is not too great. At any rate, all communication with the rest of the system should be concise.

With regard to the information needed to deal with an error situation, errors seem to be divided into two classes. The first class

might be termed 'local' errors. These are errors that are readily detected by the arm alone (error in position, etc.) and that can be corrected without regard to the rest of the environment, or the purpose of the action. A local error can be corrected by making provisions in the control procedures.

A good example of a local error is a drift in position arising during a motion of the arm from point A to point B. While performing the motion the arm has wandered off of its intended course, but it has not hit anything. To correct the error the controller need only plot a new course to point B. Dealing with drift and similar problems is clearly a minimal requirement of the system.

The more interesting class of errors, which might be termed 'global' errors, is typified by the following, seemingly simple, example. The arm strikes an obstacle. This error cannot be fixed without more global considerations. The obstacle cannot be removed unless the arm can determine that the obstacle is not needed where it is. The arm cannot just go around it, because the obstacle may have been damaged. The arm cannot even tell this without the vision system. If the obstacle was damaged, it must be fixed.

In general if the arm system is going to be able to deal with global errors, it must be able to decide on a course of corrective action based on the information available to it. It is clear that if the arm system is going to make intelligent decisions to correct such problems, even in the simple case above, it must have access to a complete model of the environment, and to a model of what the overall

system is trying to achieve. In addition it needs to be able to use the vision system for inspection of the environment.

It is also clear that the arm system cannot maintain either of these models alone. They require inputs mainly from the other parts of the robot system. Thus very high speed communication would be needed on all of the links suggested in Fig 2.

Further, since at a higher level there must already be a program capable of such intelligence, why not use it? Why not take advantage of the wide gulf between the two error classes and not implement any of the dotted communication links in Fig. 2? It was decided that the most intelligent thing the local arm system could do when it encounters an unanticipated error is to call for help, trying to be specific about the error that happened, and about what it was trying to do at the time, and to request further instructions from above.

III.3 SOME EXAMPLE SCENERIOS

In order to further clarify the above issues, scenerios of four typical arm actions are given. For each scenerio the paper indicates:

1) What information is needed to initiate the action.

2) What additional information must be developed by the arm controller in order to perform the action.

3) Some of the local errors the arm controller can correct itself.

4) Some of the errors it is not expected to handle.

III.3.1 MOVE TO POINT B

This first action is perhaps the most basic arm motion. The arm is at point A, and it is asked to move along a smooth path to point B without hitting anything. Only point B needs to be specified in order to start the action. The arm controller has to do a lot of work:

a) It has to decide on an exact path from A to B that the arm is capable of following without any high acceleration which would make the motion jerky.

b) The system should also have some basic ideas about object avoidance. For instance, if A and B are both on the table, and far apart, then the arm controller could plan a path that comes up off of the table in an arc, not a path that skims over the table. Here is a place that a model of the environment would be very useful, but how much model is enough and how hard would it be to keep accurate? The problem can be avoided if the higher level programs have a good model of the arm, and how the arm controller plans trajectories. If the higher level concludes that a (MOVE_TO B) command would cause the arm to hit something then it just needs to break the move into two parts (MOVE_TO C) (MOVE_TO B) which avoid the obstacle. It seems that the simple heuristic of moving up in an arc on a motion will avoid most collisions.

Turning to error situations, the main error the controller is expected to be able to deal with is the drift discussed above. In short unless the arm strikes something, the controller will get it to B one way or another. Note that if position B is not a possible configuration of the arm, the controller will detect this through its model of the

actual arm, and complain to the higher level.

The arm controller is not expected to handle the problem of striking something on the way to B. In general it has to appeal to higher authority in order to determine if the arm has done any damage. It could have a criterion, such that if a collision was soft enough it would assume that the object was undamaged, and then try to avoid it and continue on as though nothing had happened

In this case the controller must be careful that it does not go into an infinite motion trying to avoid an object. A particularly pathalogical case occurs if point B is in an object.

III.3.2 PUSH A BLOCK ASIDE

Here is another rather basic motion. A block with dimensions D is at position A and the arm is asked to push it a distance d in direction R. The request must specify A, d, R, and at least some information about D, in order to start the action.

The arm controller is going to have to figure out:

a) How the hand is to approach the block. In general, it should be moving along R towards the center of mass of the block (point A) with the longest axis of the hand perpendicular to the motion (so that the block will not rotate when contact is made with the hand, and will move in direction R).

b) It must also decide how much force to apply to the block. Again the local errors the controller can deal with are drift-like.

a) There is drift in the motions.

b) In addition several problems can arise which are very similar to drift. For instance if the block sticks, the arm tries a little more force, if the block skates along it tries a little less force. More interesting, if the block slides off of the hand, and looses contact, then the arm must back up a little move over and try again (as long as the arm does not move too far afield). The arm controller can deal with each of these problems by simply doing what it was doing before with some parameters updated.

The kind of global errors the controller cannot handle are typified by:

a) The arm hits something.

b) The block hits something and stops moving (there is a sudden large rise in force).

c) When the arm gets to A it does not find any block, (the arm controller must be careful not to push the wrong thing).

III.3.3 PUTTING A RESISTOR'S LEADS INTO A CIRCUIT BOARD

This is a much more complex action. Let us assume that the leads are already bent and cut. The arm is asked to pick up a resistor and insert the two leads into the proper two holes. Here the position (and orientation) of the resistor, and the positions of the holes need to be specified.

This is a task requiring considerable precision of the arm system. First it must pick up the resistor very accuratly so that it will have a reasonable idea of the position of the ends of the leads. Then it must

move the end of one lead to the first hole and insert it. Finally it must get the other lead in the other hole. All through this the arm controller must decide how fast to move, how hard to push.

Some of the errors the arm can handle in this action are so expected, they will probably always happen. In particular, the arm is probably not so precise that it can just put a lead in a hole. However, the arm should be able to get the lead near the hole. (If it does not, the lead may end up in the wrong hole.) The arm must drag the lead over the board trying to find the hole. It should not do this for too long, since the hole might be missing, or plugged. In either case the arm must ask for help from above. It cannot even tell what went wrong; is the hole missing or did the arm miss it?

This whole procedure could be simplified if the arm system could ask specific questions of the vision system (such as: "where is the end of the lead?") and get a quick answer. This is the one place where it appears that a small increment in communication could yield important gains in the power of the arm system. For most problems, total communication is needed and partial communication is not helpful.

Here are a couple of other errors the arm system could not handle without extensive communication:

a) If the arm drops the resistor it is probably not going to be able to find it. It might pick up the wrong thing.

b) While trying to locate the hole, one of the leads might bend. The arm system probably would not even detect this, except through finding itself unable to get the leads in the board.

This points up the important fact that while the arm system is working, the rest of the robot system should be looking on, even if there is no direct feedback, so that it can detect errors the arm would miss. Also the eye must check that things were done correctly after the arm is finished.

III.3.4 TIGHTENING A NUT WITH A WRENCH

It is possible that if the arm was strong enough it would not need to use a wrench to tighten a nut, but let us assume it did need one. There is a wrench at location A, and a bolt at location B with a nut already started and partly tightened with the hand (this is another difficult task). The action is a repeated cycle of putting the wrench on the nut (this is the hard part), swinging the wrench through an arc, and taking the wrench off the nut. The action stops when the nut is tight.

To start the action the request must specify the position of the nut and bolt and of the wrench. An interesting variation would be for the request to specify the size of the nut, and have the arm controller remember where the tools are. This view looks at the tools as part of the arm.

As always the arm system must develop specific trajectories for all of the motions involved. Here are some points of particular interest in this action:

a) When the arm picks up the wrench it must pick it up in a very precise manner, so that it will know where the end of the wrench is. In

general the arm system should have a good model of the arm wrench combination, so that it can perform taskes like putting the wrench on the nut easily.

b) The next difficult point is putting the wrench on the nut. People often do this with two hands. In order to do it with one, the arm will have to be able to accurately rotate the mouth of the wrench about the center of the nut, waiting for the wrench to slip on.

A special tool like a socket wrench or a nut driver would simplify the scenerio because the arm would only have to put the wrench on the nut once.

c) Next force feedback is used to tighten the nut by swinging the wrench. The arc the wrench can be swung through will usually be constrained by obstacles, leaving a pie shaped region in which the wrench can operate. This region must be at least 60 degrees in size for a typical wrench, and should be specified in the initial request.

d) An interesting case of local error which arises in this task is when the wrench pops off of the nut. The arm just puts it back on and continues.

e) It is interesting to note that the arm cannot tell whether it was successful in tightening a nut, or whether the bolt is cross threaded.

In all these tasks, the arm controller can only correct local errors which do not interact with anything else it is trying to do, unless the arm has access to accurate models of what is going on.

IV. THE INTERNAL STRUCTURE OF THE ARM CONTROLLER

Section II gave a basic idea of what the arm controller is expected to do. The next three sections will describe how these results can be obtained. To begin with let us summarize what the inputs and outputs of the arm hardware are.

A) Inputs from arm controller (direct effects of controller)
   1) Motor torques (at each joint (6))
   2) Joint lockers (at each joint (6))
   3) Hand control inputs (open close)
B) Output to controller (observables)
   1) Internal sensors
      a) Joint positions (at each joint (6))
      b) Joint velocities (maybe)
   2) External sensors
      a) Force sensors in wrist
      b) Touch sensors on hand
C) Ouputs to the environment (direct effects of arm)
   1) Motion through environment
   2) Forces applied to objects in the environment
D) Inputs from environment (external sensation)
   1) Due to contact with objects in the environment
      a) Force sensation
      b) Touch sensation

Fig. 3 Input/output description of the arm hardware

IV.1 ASV, ACV, AND G

What the arm controller is trying to control is the output to the environment (see Fig. 3). Due to the decision to isolate the arm system, the only way the controller has of monitoring this output is through the arm's own sensors (apart from some possible hand-eye coordination).

Fortunately the arm's sensors are often sufficient for observing the outputs to the environment. However, there is one important point

which cannot be ignored. That is the question of slop and inaccuracy in the arm. The arm controller's model of the arm and its abilities is only approximate, and the sensors have limited accuracy. This is the major cause of drift in the arm's actions. This drift can be corrected, but only within certain limits. There is a definite upper limit to the precision of control possible.

Returning to the main discussion, as a notational convenience, the ARM STATE VECTOR (ASV) formed mainly of the sensor outputs from the arm to the controller (i.e. the observables) is introduced.

ASV = (position, velocity, force, touch, hand, lockers; time)

This vector contains all of the information that the arm controller has about the state of the arm without getting additional information from the rest of the robot system. The information in the ASV is directly as it comes from the sensors. It is in the generalized coordinates appropriate to the arm (i.e. the joint angles and velocities).

Similarly the ARM CONTROL VECTOR (ACV) which gathers together the control inputs to the arm (i.e. the direct effects of the controller on the arm) is introduced.

ACV = (torques, joint lockers, hand control; time)

Note that these two vectors are not unrelated.

ASV(T+dT) = F(ASV(T), ACV(T), ENV(T))

That is, the new state is a function of the old state, the control inputs, and the environment. In order to control the arm, the controller attempts an inversion of this relation.

ACV(T) = G(ASV(T), ASV(T+dT), ENV(T))

Unfortunately this inversion can only be partial for several reasons:

a) As mentioned above, the controller has very little knowledge of the environment (ENV(T)). Therefore it continually runs the risk of encountering unexpected difficulties. Each command to perform an action gives the minimal description of the environment that is necessary for the execution of the action, but no more.

b) A less important problem is that G is not single valued. The controller must often choose between several ways of making a state transition.

c) More unfortunately, G is not a total function. Many (in fact most) state transitions are not possible at all. Intuitively this is clear because while the ASV has 18 independently varying major components (6 positions, 6 velocities and 6 forces), the ACV has only 6 independently varying major components (the 6 torques). Thus in general the controller can only control a 6-dimensional subspace of the ASV in one time interval.

d) Lastly, even when G is computable, it can be computed only approximately due to the great mathematical complexity. This is another major source of drift in the arm's actions.

Stepping back a moment, consider that the input to the controller from the robot system is a high level command such as (MOVE_TO B), while the input to the arm is summarized in ACV(T). This suggests a two stage controller, with one stage that transforms the high level command into ASV(T+dT) and the other that computes ACV(T) by using G.

IV.2 TWO STAGE CONTROLLER

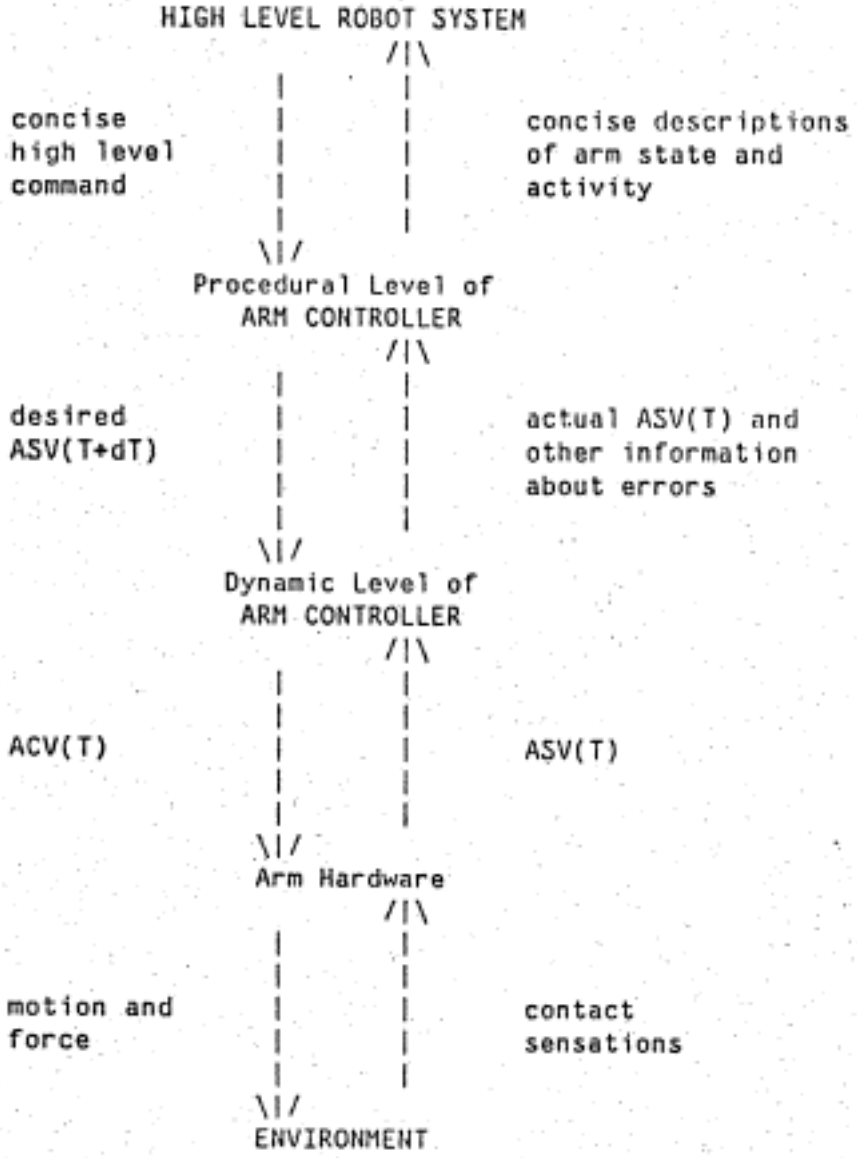

Fig. 4 A diagram of the two stage arm controller showing the information communicated between levels.

The belief that having a two stage controller will be very

convenient is based on the following assumptions on the interactions of the two stages depicted in Fig. 4.

a) First, it is assumed that the procedural level which transforms a high level command into a series of states of the arm does not need to know how the state transitions can be realized.

b) Second, it is assumed that when the dynamic level is evaluating G, in order to calculate how to get from state A to state B, it does not need to know why the procedural level wants to make the state transition or where the arm is going next.

It seems that in general these assumptions are valid. There are several clear advantages to the two stage design.

a) The computation of G is very complex, and elaborate approximations must be used. Thus it is very important to isolate the calculation of G in one place where it can be done with the greatest possible efficiency.

b) The division also makes the system clearer and more tractable. The dynamic level can be looked at as an emulator which makes the arm hardware look like it takes desired ASVs as inputs instead of ACVs. Thus from the point of view of the procedural level, the direct effects have been improved. It never has to deal with the complex mathematics in G. Nor does it ever have to deal with the generalized arm coordinates. It can work in coordinates suited to the action, not the arm.

c) Another important point involves the 'intelligence' of the mini-robot as a whole. In order to 'understand' what it is doing, it

must have some way of illucidating the inner structure of its actions. But how far should this decomposition be allowed to progress?

With the two stage controller, the answer is clear. The dynamic level operates as a black box, the complex mathematical knowledge in it forever hidden from the rest of the system. The procedural level on the other hand has all of its knowledge symbolically represented in procedures which are composed of named primitive procedures.

Thus the mini-robot can introspect down to the level of these primitive procedures, and no farther. This closely parallels the human situation. A person can introspect about his motions down to a certain level, but no farther. For example, most people think of wiggling their ears as a primitive action (even though it is produced by the combined action of many muscles), because they are powerless to control these muscles below a certain level. Most people can wiggle both ears, but very few can wiggle just one, even though there are seperate groups of muscles on each side. This is because they cannot decompose the action. They have no 'names' for its constituents. No way to get a hold on them.

d) A final point is that some time in the future, the entire dynamic level could be implemented in hardware, increasing its speed by switching to analog methods of computing G.

V. THE DYNAMIC LEVEL OF THE ARM CONTROL SYSTEM

A) Outputs to the arm hardware
   1) ACV(T) the control inputs
B) Inputs from the arm hardware
   1) ASV(T) the current state of the arm
C) Inputs from the procedural level
   1) The desired ASV (DASV) at the next time interval DASV(T+dT)
D) Outputs to the procedural level
   1) ASV(T)
   2) DELTA(T) this is a concise statement of the difference between ASV(T) and DASV(T). It is expected that they will not be the same because the dynamic level's evaluation of G is only approximate, and it has almost no knowledge of ENV(T).
   3) When an error condition arises
      a) The procedural level is interrupted and
      b) Passed a concise description of the error. (Note that this exactly parallels the interaction of the arm controller as a whole with the rest of the mini-robot system.)

Fig. 5 input/output description of the dynamic level.

V.1 THE DASV AND CONTROL STRATEGIES

As mentioned above, the dynamic level cannot control all of the variables in the ASV at once. In recognition of this, the format of DASV(T) is designed so that the procedural level can indicate which components of the ASV it is really interested in controlling. The DASV has the same components as the ASV, but each one can be either:

a) A specific number, which indicates that this component is to be explicitly controlled.

b) A list (NOMINAL_VALUE, RANGE), which indicates that an error should be signaled if the component gets outside of the interval (NOMINAL_VALUE +- RANGE).

c) NIL, this indicates that the procedural level is not concerned with the value of this component.

Based on the form of the DASV, the dynamic level uses one of the following basic strategies to compute G.

a) If the DASV indicates that position and/or velocity is to be controlled, then the equations of motion for the arm (actually an approximation to them) are used to compute the ACV.

b) If the DASV indicates that force is to be controlled, then direct feedback is used to control it. How the motor torques effect the forces sensed must also be calculated. This depends on the arm position and the point of contact.

c) If less than a 6 dimensional set of components is specified for control (in particular if none are specified), then the extra freedom of choice for ACV(T) is used to keep components of DASV(T) specified as intervals closer to their nominal values. An example of this is in pushing a block aside. The DASVs have no specific components. Before the hand touches the block, it moves at the nominal velocity. After contact, if the force rises above the nominal value, then the velocity will drop.

d) The preceding was an example of a situation where contol strategies a) and b) had to be mixed. This is done by applying both, and using the extra freedom of choice in each one to make the two solutions fit. For example if 4 positions were specified then, this would fix 4 joints. If in addition 2 forces were specified, then the 2 remaining joints would have to be used to create the forces. If this

could not be done, the dynamic level would complain.

e) Note that no attempt is made to control the touch sensations. Their state depends on ENV(T) which is unknown to this level. There is no way that the sensors can be made to turn on in a short time dT. The dynamic level just reports their value, and can be set to cause an interrupt when they change state.

f) At the opposite extreme, the dynamic level regulates the state of the joint lockers and hand. Thus an error in these is never detected.

V.2 THE BASIC ALGORITHM OF THE DYNAMIC LEVEL

The dynamic level is activated at a discrete set of times Ti, in order to produce a step function approximation ACV(Ti) to the actual ACV(T) which would produce the desired action. The Ti are specified in the time component of the DASVs. The closer together they are, the better the approximation is. How good an approximation is needed depends on the precision of the action.

At each time interval the dynamic level executes the following algorithm:

1) Obtain ASV(Ti) from the arm sensors.

2) Compute DELTA(Ti) from ASV(Ti) and DASV(Ti) as follows:

for each element of DASV(Ti) one of three cases applies

a) component absent, DELTA(Ti) = ASV(Ti)

b) component an interval, DELTA(Ti) = ASV(Ti) - (NOMINAL_VALUE(DASV(Ti))). In addition, if ASV(Ti) is

outside of the allowed interval then interrupt the procedural level and inform it of the error.

c) component a number. DELTA(Ti) = ASV(Ti) - DASV(Ti) (note that the probability is near zero that DELTA(Ti)=0, even though this is the goal. However, the dynamic level takes no action here no matter what DELTA(Ti) is. Rather it waits until step 4). That is, no matter how bad it did last interval, it will go on if it thinks that it can get to where it should be at the next time interval. In any case, it can use DELTA(Ti) to improve its approximation to G.)

3) get DASV(Ti+1). The procedural level communicates with the dynamic level through a queue of DASVs. If this queue is empty, the dynamic level interrupts the procedural level and requests a DASV. The arm must be continually controlled. Until the dynamic level gets a DASV it will continue the current ACV with possibly bad results.

How close the coupling between the two levels is depends on the length of the queue. The procedural level can take complete control by supplying the DASVs one at a time. Alternately the procedural level can leave the dynamic level alone indefinitely by putting a loop in the queue.

4) Calculate ACV(Ti) and apply it to the arm. To do this, the dynamic level must first check to see that DASV(Ti+1)

is achievable. If not, it interrupts the procedural level and requests a new DASV(Ti) which is achievable. Also note that some subcomputations (such as evaluation of the equations of motion) are very time consuming and will have to be spread over several time intervals.

5) Wait until Ti+1 and then go to 1). It is very important that the dynamic level begin its computations at time Ti+1; it cannot wait for anything. Also the entire computation must be short compared with the length of the time interval, or the control will deteriorate. To this end, it is clear that the arm system as a whole will have to run as a set of asynchronous tasks sharing the CPU time.

V.3 ERROR DETECTION IN THE DYNAMIC LEVEL

Let us take a final look at the 'error detection' the dynamic level is doing. It is of two types:

a) The dynamic level complains because it cannot make a requested state transition (step 4). The problem could have been caused by one of two things:

1) An impossible sequence of DASVs was issued. This should not occur if the procedural level has a good enough model of the arm's abilities.

2) Inaccuracies in the computation of G and in the arm hardware caused the dynamic level to wander too far from the requested trajectory for it to recover.

b) The dynamic level complained because some element of DASV(Ti) (expressed by a range) was not consistent with ASV(Ti). Essentially these range elements in the DASV are used to hold predictions of the corresponding elements in the ASV. The dynamic level simply informs the procedural level whenever the procedural level's prediction is incorrect. (This basic method of error detection was used by Ernst).

Since the dynamic level knows nothing about the 'purpose' of its actions, or about ENV(T), it really does not know what is happening. It knows that situations of type "a" (above) are errors. However, it does not know what the 'meaning' of a type "b" situation is. The interrupt mechanism is just used so that the procedural level can request the dynamic level to bring key points of information to the attention of the procedural level. The procedural level must decide what the information 'means'.

For example the information that a touch sensor has changed state could mean "the hand just crashed into the table" or it could mean "the hand just found the capacitor". It all depends on what is being done.

The above mechanism is probably not sufficient for all error detection. In addition there will probably be 'demons' in the procedural level that monitor more complex functions of the ASV or other indicators in order to detect error situations. The interrupt mechanism merely institutionalizes the most common kind of monitoring.

Also one element is added to the DASV not found in the ASV. This element simply causes an interrupt whenever it is present. This is used to synchronize the actions of the dynamic and procedural levels. In

particular it would be used to inform the procedural level that a particular phase of on action was completed.

In summary then, the dynamic level is seen to be quite simple, except for the mathematical complexity needed to carry out the control (i.e. computing G).

VI. THE PROCEDURAL LEVEL OF THE ARM CONTROL SYSTEM

A) Output to the dynamic level
   1) DASV(Ti) the desired ASV (communicated through a queue)
B) Inputs from the dynamic level
   1) ASV(Ti) the current state of the arm
   2) DELTA(Ti) the summary of differences between ASV(Ti) and DASV(Ti)
   3) Interrupts signalling errors and key information detailed in DELTA(Ti)
C) Inputs from the high level robot system
   1) High level commands in the form of LISP programs
   2) Interrupts signalling error conditions or key information detected by other elements of the mini-robot system (for example the vision system)
D) Outputs to the high level robot system
   1) Signals indicating the completion of an action
   2) ASV(Ti) (in some high level coordinates) if requested
   3) Interrupts signalling error situations
   4) A description of any error (see V.3 below)

Fig. 6 Input/output description of the procedural level

VI.1 THE PROCEDURAL LEVEL IS A LISP SYSTEM

The procedural level is a LISP system with a large body of built-in functions. These functions can be divided into three classes:

a) The first class consists of the basic LISP functions (PROG, LAMBDA, COND, etc.) plus a set of functions to implement the multiple asynchronous control paths necessitated by the real-time demands on the system. These include functions for synchronizing communication between, creating, and deleting control paths.

In addition there are functions for creating and fielding interrupts (see V.2 below).

b) The second group of functions embody mathematical information

about the arm and its capabilities. These include functions for transforming between coordinate systems, planning trajectories, etc. The dynamic level can be thought of as one of these programs. These programs are implemented in machine code for efficiency.

c) The last class is the most interesting. It is the set of procedures for performing actions. These programs are kept as lists, and are available for inspection by the higher levels. The commands sent by the high level system are either direct calls on these procedures, or more complex procedures built out of the functions already in the system. Procedures can be added, deleted or altered by the higher level system in order to suit its needs. Some examples of basic procedures are:

1) MOVE_TO B.

2) HALT arm where it is.

3) APPLY_FORCE F to point A.

4) MOVE_IN_DIRECTION R from point A until contact is made with an object, then halt.

5) GRASP the object between the hand's fingers (some routine called by this routine must calculate the change in the inertial description of the arm when it picks up the object).

Some examples of more complex procedures that might be in the system if it were working in the environment of circuit board construction are:

1) INSERT_RESISTOR at A in holes at B.

2) START_NUT at A on bolt at B.

3) SOLDER_LEAD in hole at A.

4) ATTACH_TEST_PROBE to lead at A.

Any action can be encoded as a procedure as long as it can proceed by dead reckoning without feedback from the rest of the system. Whether procedures like 1-4 above will be done with or without visual feedback depends on the taste of the users, and the speed of visual processing.

VI.2 HANDLING ERROR SITUATIONS

It is clear how everything works as long as no errors arise. However, it is the error situations that are the most interesting.

Information about errors is brought to the attention of procedures able to deal with them through interrupts issued by other procedures which have gotten into trouble. The interrupt facility is basically a simple pattern-directed invocation facility with two key differences.

a) It is assumed that only one procedure will be appropriate at a time. This assumption does not seem unreasonable in light of the simple kinds of errors with which the arm control system is trying to deal.

b) Control is almost never returned to the routine issuing the interrupt. The issuer is usually supplanted by an updated approach to the problem facing the arm control system.

The interrupt mechanism is implemented by two functions:

a) (SIGNAL INT ADDED_INFORMATION) Where INT is an atom (the pattern for pattern-directed invocation), and ADDED_INFORMATION is anything that the signaller feels will be useful to the invoked procedure.

b) (ON INT PRED ACTION) When an ON is executed, it enters the triple (INT PRED ACTION) on a stack. The triple is removed from the stack when the PROG the ON was executed in is exited.

When an interrupt is signalled, the stack is searched for an appropriate triple. A triple is appropriate if INT is the same as the one signalled, and PRED is true. When an appropriate triple is found, ACTION is evaluated in the environment of the PROG in which the ON was executed.

Here is a simple example of an action procedure using these ideas.

```
(DEFUN MOVE_TO (B)
      (PROG ()
            (ON COMPLETION T (RETURN))
            (ON UNABLE_TO_CONTINUE_MOTION T (GO RETRY))
      RETRY (SETQ DASVQ (MAKE_TRAJECTORY B))
            (SUSPEND) ))
```

In the above, DASVQ is the queue of DASVs used to communicate with the dynamic level. COMPLETION is the interrupt that is generated by the additional element of the last DASV in the queue. UNABLE_TO_CONTINUE_MOTION is the interrupt signalled by step 4) of the algorithm running in the dynamic level. MAKE_TRAJECTORY is a mathematical routine that plans a trajectory from the present location to the point B. The returned queue of DASVs is terminated by one set to signal COMPLETION, and one that points to itself to keep the arm stationary and waiting at point B. SUSPEND is a function that stops execution in a control path. Execution can be resumed by the action of

another control path.

The structure of this program is typical. A procedure usually sets up a series of alternatives for future action, computes some number of DASVs, and then suspends itself. It will later be reexcited by an interrupt from the dynamic level.

V.3 A DEEPER LOOK AT ERROR SITUATIONS

The interrupt facility is intended to give the procedural level the ability to deal with simple local errors, that is the errors described in section II as being easy enough for the arm control system to deal with alone.

But what about difficult errors? How are they detected and what does the procedural level do about them? The only way the procedural level has of observing errors is through the ASV. As long as it contains values consistent with there being no errors, then no error can be detected by the procedural level. As soon as the ASV fails to agree with the prediction of the procedural level (or a watching demon sees something wrong) an interrupt is issued. At this time, the ON stack is searched for an approriate triple. If one is found, fine. If one is not found, then the procedural level concludes that it cannot deal with the error. At this point, the high level system is interrupted. Also since it may be a long time before the high level system responds, the arm is told to halt where it is, and the whole arm control system waits for a command from the high level system.

Note that the interrupt might not have originated in the arm

control system, but rather in some other component of the mini-robot system. For instance, the vision system might have detected something which it wished to report directly to the arm controller rather than to the high level system. If the arm controller does not know what to do about it, it will just reflect the interrupt up to the high level system.

In summary, from the point of view of the procedural level, a difficult error is an interrupt for which no specific ON has been executed. Whenever a difficult error occurs, the procedural level asks for help and waits.

Now the important question is: how is the high level system is going to respond to an error interrupt? There are two main ways for it to proceed.

a) The clean slate approach: Here the high level system decides not to try and salvage anything of the current state of the arm control system. Rather it just looks (with vision etc.) at the current state of the environment, and on the basis of high level goals creates an entirely new procedure to send to the arm control system, in order correct the error and continue on.

b) The patch up approach: Here the high level system tries to patch up the partially executed procedures in the arm controller in order to correct the error, and then has them continue on. This has the advantage that partial computations are not lost. Also there is a good chance that similar patches can be given to similar errors even when they occur in the context of very different actions. The biggest

disadvantage of this appraoch is that the high level system must know a great deal of information about the arm control system in general, and about the specific control paths currently being executed, in order to make a good patch.

Also note that the second approach is not able to do anything that cannot be done with the first method. The second method is just more convenient. Further, the gain is greatest if the ongoing process is complex. If it is simple then the high level system might just as well start over.

When the procedural level gets an error interrupt from the dynamic level, it could also use either the clean slate or patch up approach. Mainly because a queue of DASVs does not really contain that much information, the first approach is usually used. A whole new queue of DASVs is created.

In the initial arm control system the high level system will use the clean slate approach. At a later date however, it will be of great interest to implement constructs that will allow the high level system to talk with the procedural level about its internal state of execution.