A VISION POTPOURRI

VISION FLASH 26

by

Tim Finin

Massachusetts Institute of Technology

Artificial Intelligence Laboratory

Robotics Section

JUNE 1972

Abstract

This paper discusses some recent changes and additions to the vision system. Among the additions are the ability to use visual feedback when trying to acurately position an object and the ability to use the arm as a sensory device. Also discussed are some ideas and a description of preliminary work on a particular sort of higher level three-dimensional reasoning.

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-0362-0003.

JIGGLING A BLOCK INTO PLACE

The vision system can now use visual feedback when trying to accurately position a block. This is done without a costly rescanning of a significant portion of the scene by using our knowledge of where the block should be to direct the eye. The basic idea is to determine the block's actual location by looking for certain key vertices using a circular-scanning vertex finder developed by Winston and Lerman < Vision Flash 24 >.

When placing a block the arm sometimes makes positional errors up to half an inch and rotational errors of about 10 degrees. These errors are caused by poor hand placement due to hysteresis and general slop in the arm's joints and by poor information about the brick's initial position and dimensions due to a distorted line drawing. Although these errors can be disastrous in delicate tasks such as stack-building, they are small enough to allow us to use the scheme described below.

The organization of the theorems is shown in figure 1. TC-JIGGLE, the top level theorem, first calls TC-FIND-BODY whose goal is finding the actual location of the just moved brick. This is done by locating a three-vertex 'skeleton' on either the top or bottom of the brick , examples of which are shown in figure 2. Candidate skeletons are suggested by the theorems TC-LOOKFOR-TOP, TC-LOOKFOR-BOTTOM, and TC-LOOKFOR-SKELETON which predict the locations of vertices and decide whether they should be visable. TC-FINDBODY then locates the three vertices comprising the skeleton with the circular-scan vertex finder and calculates the true position of the brick. If it fails to find one of the vertices, it asks for another skeleton and tries again.

Once the location of the brick is found, TC-SHIFT-BODY calculates the positional and rotational errors and, if they are greater than a tolerance, corrects them thru a call to TC-MOVE-GENTLY. This theorem differs from the usual TC-MOVE in calling the arm with GRASP and UNGRASP commands instead of PICKUP and DROP. PICKUP and DROP raise the arm several feet above the table when moving to avoid obstacles, whereas GRASP and UGRASP lift the hand less than an inch (using the wrist) and thus, hopefully, are less prone to error.

The most difficult part of this jiggling procedure is determining which vertices of a brick will be visable and not obscured by other objects. We must also avoid looking for vertices which are adjacent to others already in the scene , for example the vertices where two bricks are aligned. Such situations may confuse the vertex finder and cause it to find the wrong vertex. Since these theorems are written to work in the context of a copying task, they use information about the model scene that is being copied. For instance, before TC-LOOKFOR-TOP looks for any vertex on the top of a brick it must either find that:

1. The top of the matching brick in the model was

completely visable.

2. All bricks which could be adjacent to the one in question are either below it or have not yet been placed.

The theorems lean toward conservatism in accepting vertices as good candidates to look for and will reject all of them in some cases.

One exciting possibility for further work is the incorporation of a model of the hand. With it we could adapt the system to avoid vertices occluded by it, doing away with the necessity to release the brick and withdraw the hand. This would result in a more dynamic and accurate feedback system.

$$\begin{tikzcd}
 & & & {} \arrow[dl] & & \\
 & & \text{TC-JIGGLE} \arrow[dl] \arrow[drr] & & & \\
 & \text{TC-FINDBODY} \arrow[r] \arrow[dl] \arrow[d] \arrow[dr] & \text{VERIFY-VERTEX} & & \text{TC-SHIFTBODY} \arrow[d] & \\
\text{TC-LOOKFOR-TOP} \arrow[dr] & \text{TC-LOOKFOR-Bottom} \arrow[d] & \text{TC-LOOKFOR-skeleton} \arrow[d] & & \text{TC-MOVE-GENTLY} \arrow[d] \arrow[dr] & \\
 & \text{TC-CLEARVIEW} \arrow[d] & \text{TC-CHECKPOINTS} \arrow[d] \arrow[dr] & & \text{GRASP} & \text{UNGRASP} \\
 & \text{TC-NEIGHBORING-bodies} & \text{TC-GOOD-CHECKPOINT} & \text{TC-SKELETON} & &
\end{tikzcd}$$

FIGURE 1

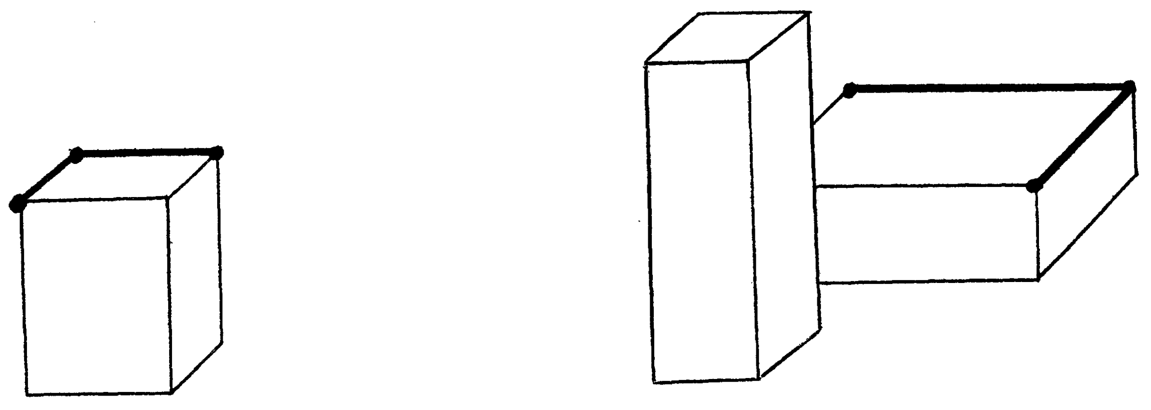


FIGURE 2

OUR ROBOT HAS A HAND, TOO

Until now the vision system has made no use of its arm in getting information about the world. We now have a limited ability to reach out and touch in order to disambiguate some scenes, using a new arm primitive written by Jerry Lerman.

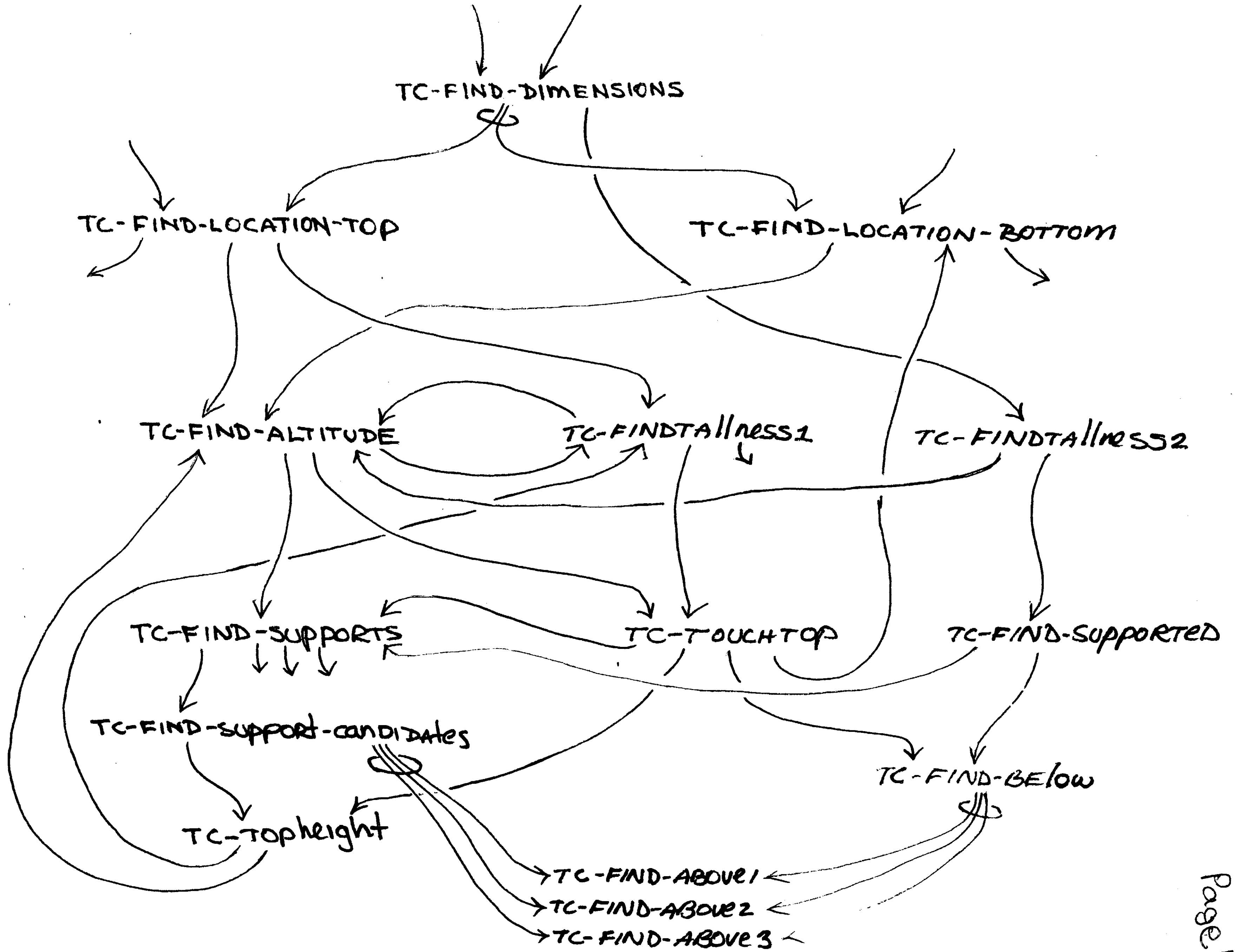
Sending (TOUCH X Y) to the arm causes it to position itself above the point (X,Y) , slowly descend until it touches something, and report its final height. An optional third argument can specify a maximum height at which something is expected, allowing the arm to rapidly drop to this height and then more slowly feel its way downward.

A series of theorems have been written which activly use the arm as a sensor and other theorems have been taught to use them, resulting in the system network shown in figure 3. With these theorems we can now handle scenes such as the pedestal in figure 4. In this scene we can't determine the tallness of B1 , since it could touch the bottom of B2 near the front, the back, or somewhere in between. As a result, we can't get the dimensions and location of B2 either.

We can however determine the location of B1 in the X-Y plane (thru TC-FIND-LOCATION-BOTTOM) . Moving the arm down over this spot until it touches the top of B2 gives the altitude of B2's top. With this information we can calculate the location and dimensions of both bricks.

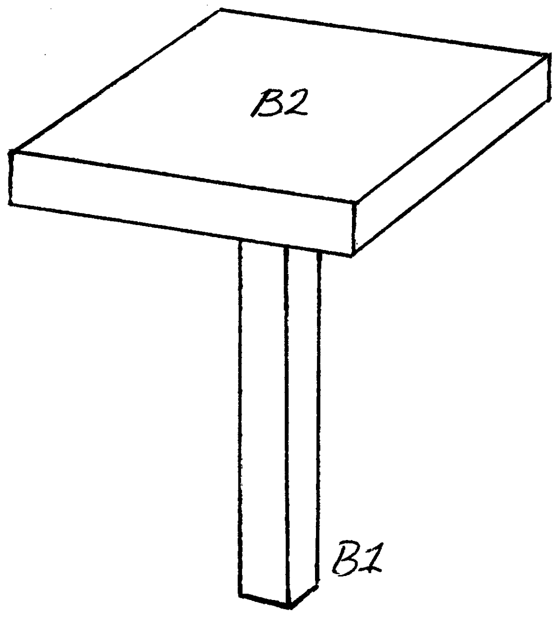
Previously, when we wanted to find the location or dimensions of a brick we had to find its altitude above the table. If it was not resting on the table, we had to find the dimensions of its supports, necessitating knowing their altitudes above the table, etc...... We recurse downward until we reach the table or fail by hitting a brick for which no tallness or altitude can be found. With these new theorems we have another alternative: recursing upward until we find a brick we can touch with the arm.

One problem is that we aren't working with a very good three dimensional model. TC-TOUCHTOP is the theorem which tries to touch the top of a brick. Checking first that there is nothing above the brick, it tries to touch it above the center of one of its supports. The brick could, however, not be above this spot (as in figure 4b) causing the arm to miss it. One precaution that TC-TOUCHTOP takes is calculating the minimum height to expect the top of the brick. If it touches something below this height, it assumes it missed.



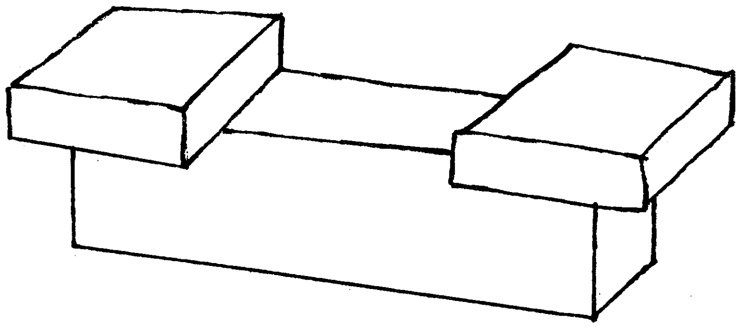
TC-FIND-DIMENSIONS
TC-FIND-LOCATION-TOP
TC-FIND-LOCATION-BOTTOM
TC-FIND-ALTITUDE
TC-FINDTAllness1
TC-FINDTAllness2
TC-FIND-SUPPORTS
TC-TOUCHTOP
TC-FIND-SUPPORTED
TC-FIND-support-candidates
TC-FIND-BElow
TC-TOPheight
TC-FIND-ABOVE1
TC-FIND-ABOVE2
TC-FIND-ABOVE3

FIGURE 3



A

FIGURE 4



B

TC-FIND-SUPPORTS

TC-FIND-SUPPORTS and its related theorems have been modified to handle situations with which they previously could not cope. Figure 5 shows the new organization of this part of the system. The strategy of TC-FIND-SUPPORTS was to take each object below the brick in question (found thru TC-FIND-ABOVE-1 & -2) as a support candidate. The altitude and tallness of each candidate were found and summed. The object or objects (if there were several with nearly equal combined altitude and tallness) with the largest sum were then taken as the actual supports. This sum was then asserted as the altitude of the supported brick. The theorem failed if it could not find an altitude or a tallness for one of the bricks below.

The new TC-FIND-SUPPORTS works in much the same way , but has been modified to handle many cases where the tallness or altitude of an support candidate can not be found. In such cases it determines the minimum height that the top of the candidate could have.

It will also yield useful information in cases where it is still ambiguous which objects support another. Before failing it makes assertions of the form:

```
(B1 may-be-supported-by B4)
(B1 may-be-supported-by B7)
(B1 has-minimum-altitude 4.12)
```

These assertions can later be used by other theorems with more real world knowledge to clarify the scene. For example, we might call on a theorem which knows about stability or one which can recognize a table top and legs to decide who is doing the supporting.

Two auxiliary theorems are used, TC-ADD-TO-SUPPORTS-1 and -2, which contain some 3-D knowledge. TC-ADD-TO-SUPPORTS-1 looks for a marrys relation between the brick in question and a support candidate. If one is found, the theorem reports that it must be a support (assuming gravity and no glue). TC-ADD-TO-SUPPORTS-2 is explained below.

The capabilities of the new TC-FIND-SUPPORTS are best shown in the scenes in figure 5 . For each of these scenes the old TC-FIND-SUPPORTS would simply fail, leaving no assertions in the data base. Figure 5e is particularly interesting , showing the application of some three dimensional reasoning. On this figure TC-FIND-SUPPORTS first calls TC-FIND-SUPPORT-CANDIDATES which reports that B2 and B3 are likely support candidates and that B1 must have an altitude of at least T. TC-ADD-TO-SUPPORTS-1 then finds that B2 marrys B1 along B1's bottom edge, implying that B2 must support B1 and that B1 has an altitude of T. TC-ADD-TO-SUPPORTS-2 is activated and notes that B1's altitude is now known to be T. Discovering that the minimum tallness of B3 is also T (within an epsilon) it asserts that B3 must also marry B1 and be a support.

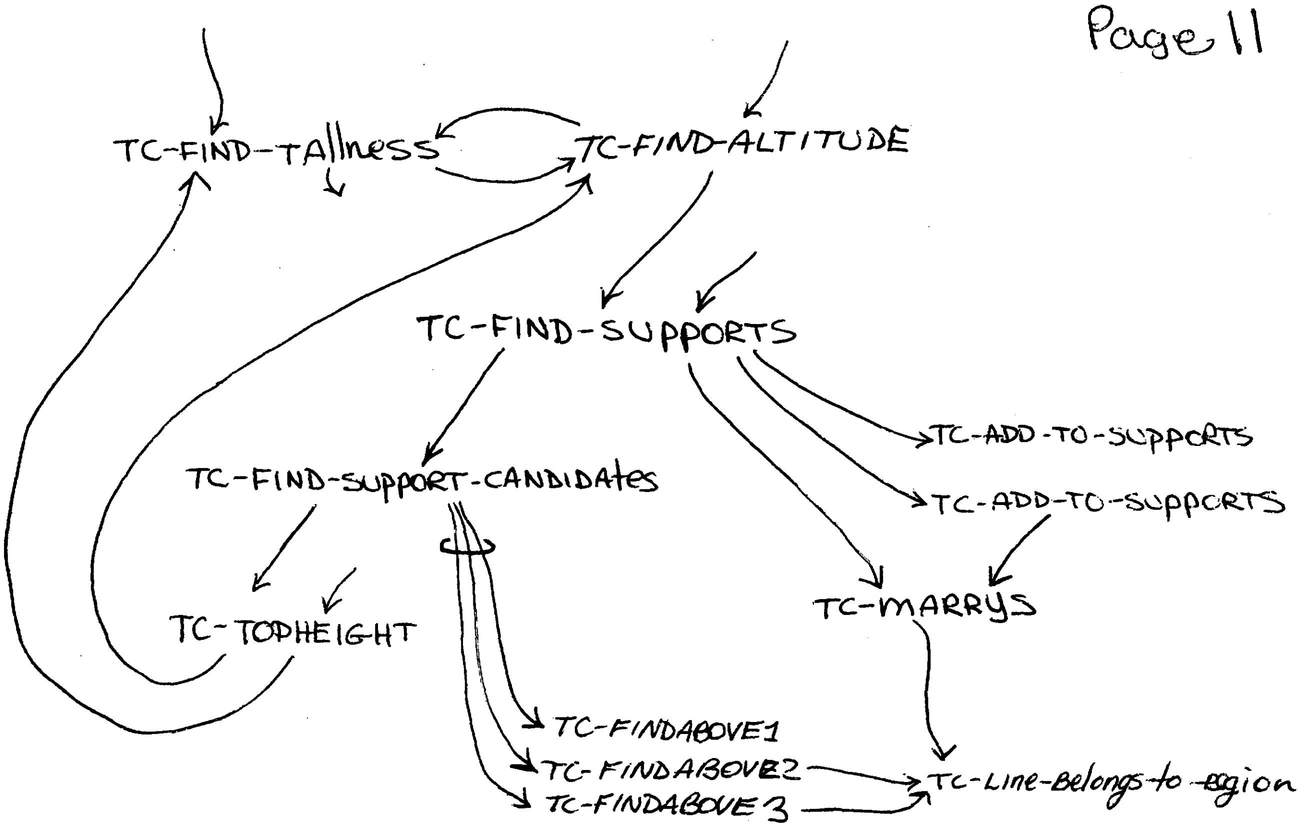
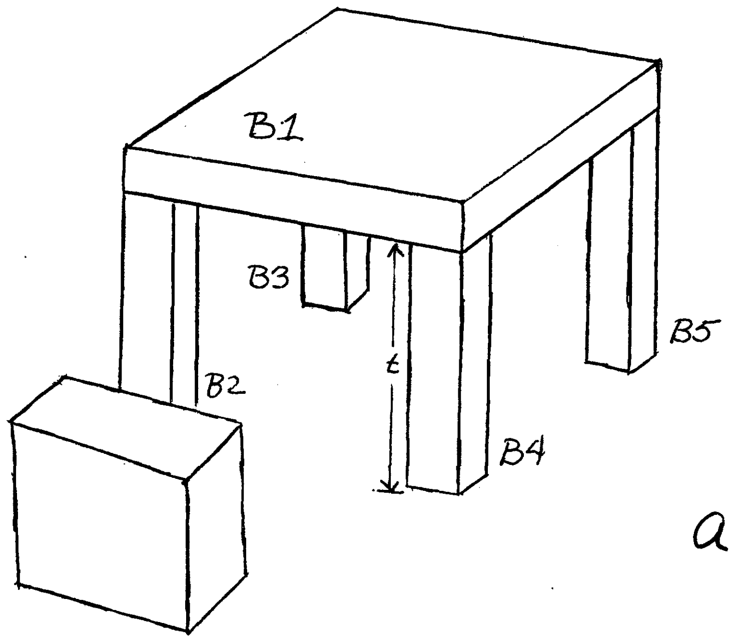


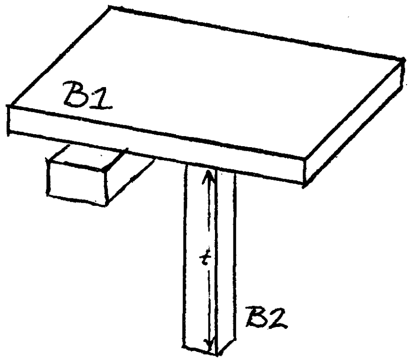
FIGURE 5



a

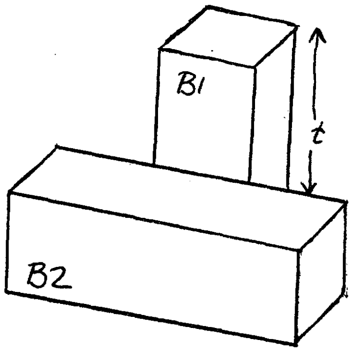
Assertions made by
TC-FIND-SUPPORTS:

(B1 is-supported-by B2)
(B1 is-supported-by B4)
(B1 is-supported-by B5)
(B1 may-be-supported-by B3)
(B1 has-altitude t)



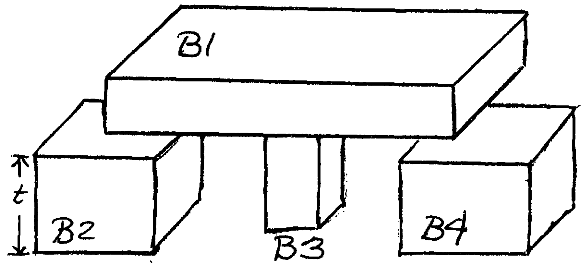
b

(B1 is-supported-by B2)
(B1 has-minimum-altitude t)



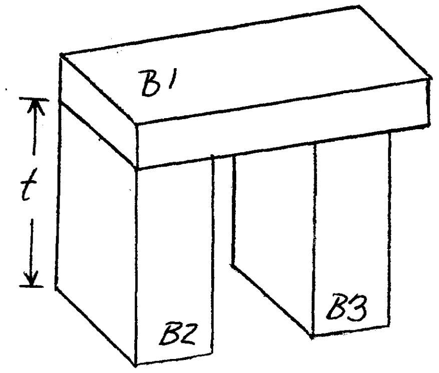
c

(B1 is-supported-by table)
(B1 has-minimum-altitude t)



d

(B1 may-be-supported-by B2)
(B1 may-be-supported-by B3)
(B1 may-be-supported-by B4)
(B1 has-minimum-altitude t)



e

(B1 is-supported-by B2)
(B1 is-supported-by B3)
(B1 has-altitude t)

CHANGES TO TC-SKELETON

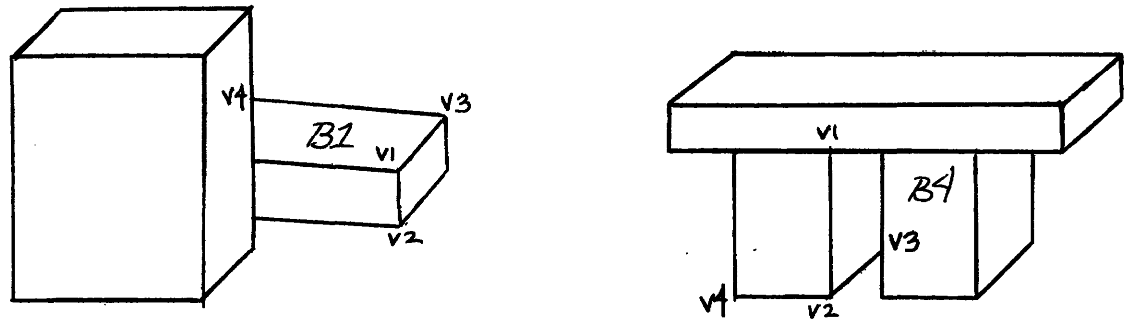
TC-SKELETON has been changed to return a little more information if it can't find a complete skeleton for a brick. When TC-SKELETON fails to find a line of a particular type it tries to find the longest line fragment of that type and makes partial-skeleton assertions of the form:

```
(B3 type-two * (V1 V2))
(B3 has-partial-skeleton V4 V1 * (V1 V2) V1 V14)
```

Figure 6 shows some examples of partial skeletons.

This partial skeleton information is used by other theorems which hypothesize what the rest of the brick may be like. From it we can get a handle on some three-dimensional information such as a brick's orientation, its minimum dimensions, and some idea of its location. Since other theorems make hypotheses that complete parts of the skeleton, an antecedent theorem has been added to keep the skeleton assertions up to date.

FIGURE 6



(B1 has-partial-skeleton V1 V2 V1 V3 * (V3 V4))

(B4 has-partial-skeleton * (V1 V2) * (V1 V3) V2 V4)

TC-LINE-BELONGS-TO-REGION

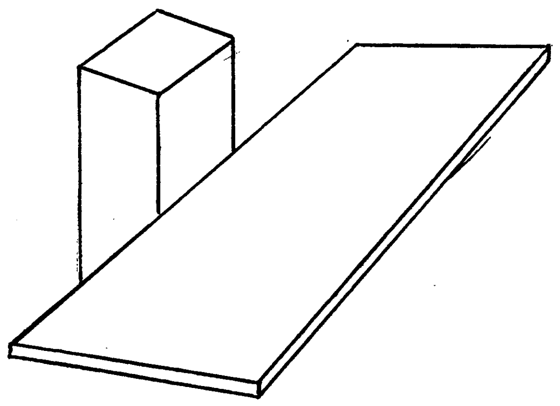
A new theorem exists which will determine whether a line is physically associated with a particular region in a picture. This question crops up in numerous places in the vision system: finding the skeleton of a brick, finding a brick's tallness, finding bottom lines, deciding whether a face of a brick is vertical or horizontal, etc. In each of these places several heuristics were employed to find lines which 'belonged' to a region. TC-LINE-BELONGS-TO-REGION is a collection of these heuristic and several new ones which can be called whenever needed. It makes assertions of the form:

(L-V1-V2 belongs-to R4)

One of the heuristics is that an interior line of a body belongs to both regions it bounds, so that TC-LINE-BELONGS-TO-REGION should not be used on self-occluding bodies. In general, the heuristics are applied conservatavely, sometimes calling on the theorem TC-OCCLUSION-1 to look for supportive or contradictory evidence. Consequently some lines will not be recognized. Having this information at hand makes many tasks much easier. For example:

* A region is a vertical face if there is a vertical line which belongs to it.
* Two objects marry if we find a line which belongs to a region from each body.

M&M HACK



How many things can you find wrong with this picture?

In a picture such as the one above where one brick obscures the bottom of another, we can extract some information on the whereabouts and size of the obscured brick. Examine the scene in figure 7 . B2 could be a very tall brick which was touching B1 , or a shorter brick far behind B1, or anywhere in between. It's ambiguous. Knowing the range of possible heights and resulting locations of the obscured brick will be quite useful to other thoerems which try to decide what the situation really is. To get quantitative three dimensional information we use the procedures described below.

To get the maximum tallness of B2 we assume that it is directly behind and touching B1. Assume for the moment that the ends of B2's three vertical edges (b, c & d) touch the edge a-e.

These three points would then have an altitude of h (the 3-D tallness of B1) from the table. From this we can get their 3-D coordinates and the resulting lengths of the three verticals. We then take the shortest of these lengths as the maximum tallness of B2. This corresponds to selecting the vertical ending in c as the only one which could touch B1.

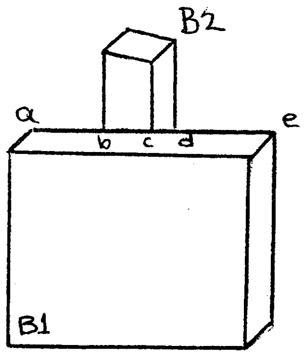
To get the minimum tallness we assume that B2 is far behind B1 and its bottom edges just barely obscured. For the moment assume all three points are on the table. We get their 3-D coordinates and calculate the lengths of the verticals. The maximum of these three lengths gives us a minimum tallness for B2. Taking the longest corresponds to selecting the point b as the only one which could actually be on the table.

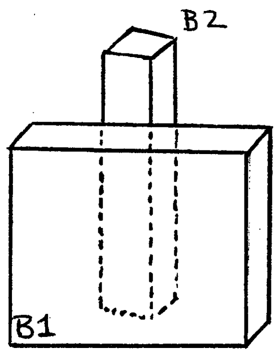
The problem with the picture on the previous page is that, assuming the obscured object is a brick, its apparent minimum tallness is greater than its apparent maximum tallness.

A further refinement of this heuristic is shown in figure 8. In this picture , to get the minimum tallness of the obscured brick we assume that the points a and b rest not on the table, but on the regions R1 and R2 respectively. We must check of course, that these regions are not vertical , as in figure 8b.

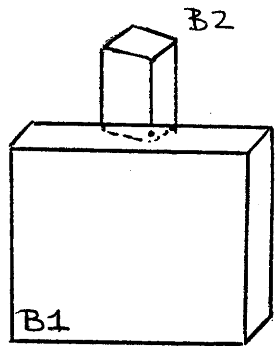
A sort of dual exists for the inverted case, the pedestal in figure 9. If we assume that B1 supports B2 , we can put upper and lower bounds on the height of B1 (and the resulting altitude of B2). We know that the top of B1 must touch the bottom of B2 somewhere - near the front, the back, or somewhere in between. Getting the minimum tallness is trivial , just measure the length of the three vertical edges of B1 and take the largest of these. This corresponds to a situation where B1 and B2 marry along their front edges. To get the maximum tallness we start off by locating the visible bottom edges of B2 and predicting where the others should be. We then extend each vertical until it intersects one of the back bottom edges of B2. After calculating the 3-D lengths of these verticals we take the shortest as being the upper bound on the height of B1.

Considering the stability of such structures would of course lead to more refined upper and lower bounds.



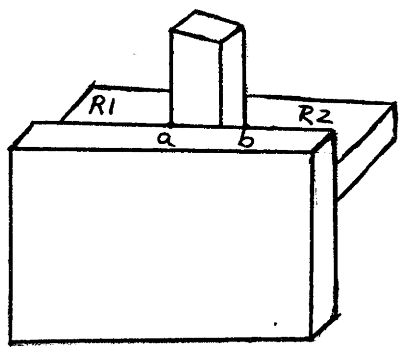


maximum tallness

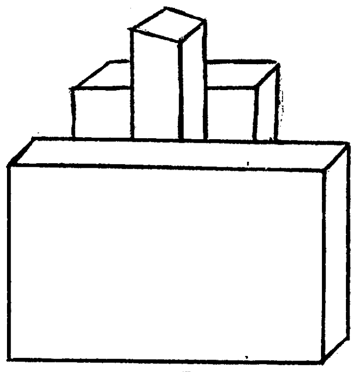


minimum tallness

FIGURE 7



A



B

FIGURE 8

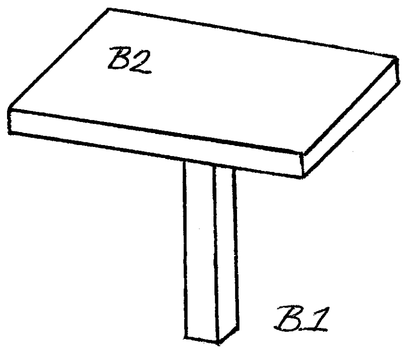


FIGURE 9

ONE KIND OF THREE-DIMENSIONAL REASONING

Everyone agrees that future advances in computer vision will come with the incorporation of 3D knowledge and reasoning. One type of 3D reasoning we could add is shown in figure 10. In figure 10a the average human viewer (trusting and non-cynical) sees two identical bricks supporting a third , even though B1 could be longer or shorter than B2. Because B1 and B2 serve the same function (supporting B3) , have the same orientation, and as far as we can see, could be the same size, we easily assume that they are. Similarly , in figure 10b , we see B4 as being the same size as the other three standing bricks. Here the evidence is:

* There is a preponderence of tall, thin standing bricks
* B1, B2, and B3 form a row, which would include B4 if it were the same size
* B1, B2, B3, and B4 all have the same orientation
* B4 could be the same size as B1, B2 and B3

Humans do this hypthesizing and filling in of details to a great degree. It is an integral part of perception, as it would be impossibly costly (in time and effort) to try to disambiguate everything we see. Teaching the machine to do such hypothesizing would be a natural way to incorporate some three dimensional reasoning and to enable it to consider the global context of the scene.

A similar form of reasoning was pointed out by Winston in his thesis < Learning Structural Descriptions from Examples , AI TR-231 >. In figure 11 B2 appears to be a wedge, while we see B4 as a brick, even though they show the same arrangement of lines and faces. In both cases since the the two objects form stacks and are exactly aligned we first assume that they are identical.

We might object that this presupposes an orderly scene , one which is carefully set up or contrived. However, the world we humans create, and which robots may inherit, is just such an orderly, contrived world. Even in the mini-world of plain white-faced blocks we use in our present research we tend to build little arches, stacks, and other stuctures containing identical, interrelated parts. In the larger world of human construction this orderliness is more apparent. We tend to build things which are symmetric and unsurprising in details. Complex objects such as buildings, electronic circuits, and cars are built using smaller identical parts (e.g. standard-sized windows, resistors, bolts). Who would suppose that the wheels on one side of a car are any different than those on the other? Long hallways usually have identically dimensioned doors uniformily spaced. The legs of a table or desk are nearly always the same.

We might also object that a robot would do better to spend his time trying to disambiguate a scene by removing some obscuring obstacles, by walking to one side, or by reaching out his hand and touching. In many cases however, our robot may find it impossible to change his viewpoint or to interact with the scene. Even more importantly, the ability to do this sort of reasoning would allow him to have some expectation as to what will most probably be seen if obscuring objects are removed, or the viewpoint is changed. The robot can then quickly test his previously formed hypothesis. If this verification fails, he can flush the hypothesis and examine the scene more carefully.

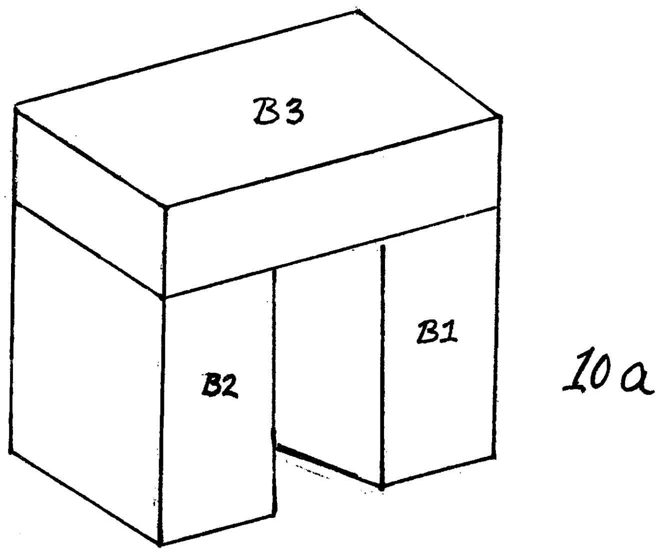
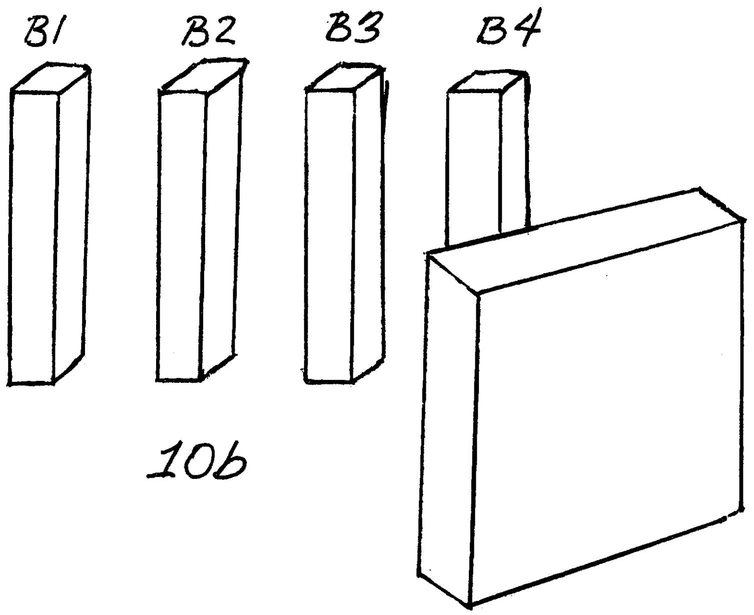


FIGURE 10



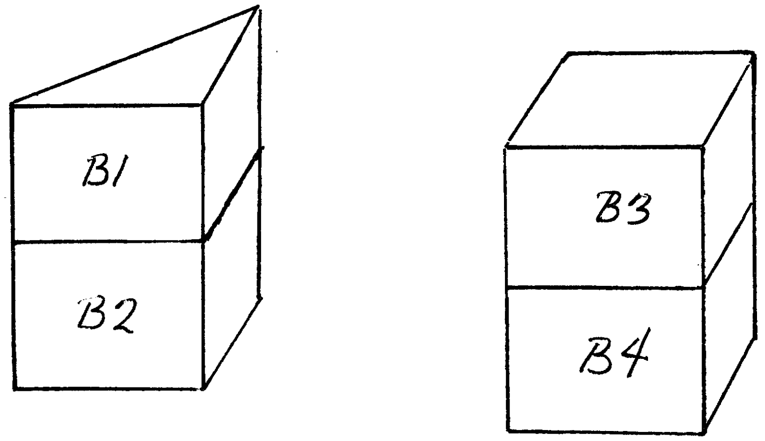


FIGURE 11

The following pages describe initial work in creating a system of theorems to do just this sort of reasoning. A skeletal system has been implemented which will handle a number of the simpler situations (such as those with fairly obvious group type relationships -- rows, stacks tables, arches, etc.).

Whenever we can't find the complete dimensions of a badly obscured object, as in figures 10a and 10b, we check for evidence that this mystery object might be just like some other object in the scene. Typical of the evidence we look for are:

* Chains of relations (rows, stacks, walls)
* Identical relations to a third object, or functional similarities (e.g. supports of an arch, table legs)
* A preponderence of identical objects in the scene
* Exact alignment with another object
* Other relations and properties such as attitude, nearness, placement, marrys, abuts, etc.

If we find such a candidate we check for contradictory evidence. Things we check are:

* That the known dimensions of the obscured object agree with the corresponding dimensions in its supposed double.
* The unknown dimension(s) lie within the

apparent maximum and minimum bounds we have calculated.

* That the hypothesis yields no colliding objects (i.e. those occupying common space).
* That we do not contradict any previous assumptions we have made (e.g. support).

In some cases we need not require an object which might be an exact double, but only one which implies a similarity that might resolve the hidden dimensions. For example, in figure 12 we hypothesize that that the two stacked bricks have the same depth , even though their heights are different.

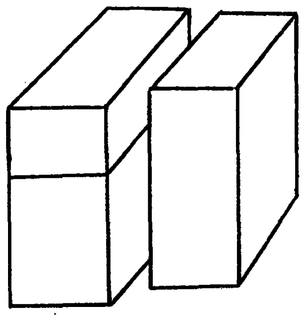


FIGURE 12

If we find any contradictory evidence, we reject the candidate and can look for another. If no contradictory evidence is found, then we make our calculations and tentatively assert the dimensions and location of the object given our hypothesis. This is done with pointers to the theorems which suggested this hypothesis so that we can reconstruct our reasoning if needed.

With this information we can proceede to analyze the scene and hope that we have not been tricked. Alternatively we can try to verify the hypothesis by some other means. For instance, we could create a daemon theorem to lurk in the background waiting for some of the obscuring objects to be removed. When they are, the eye can be asked to verify critical lines or vertices suggested by our hypothesis. For our paradigm arch case, as soon as the top brick is removed, we can look for the top back vertices of the mystery brick. If the vertices aren't found near their proposed locations then the hypothesis is flushed. In such a case we could use the eye to more completely scan the area to resolve the problem. In other situations we might be able to use the hand as a sensor to verify the hypothesis or find out what has gone wrong.