A Prototype Inference Engine for Rule-Based Geometric Reasoning

Joseph J. Pfeiffer, Jr.

Department of Computer Science
New Mexico State University
Las Cruces, NM 88003 USA
pfeiffer@cs.nmsu.edu

Abstract. Isaac is a rule-based visual language for mobile robots using evidential reasoning and a fuzzy inference engine. A prototype inference engine for Isaac has been implemented, permitting experiments with the Isaac language. This paper discusses this inference engine, describes some preliminary experiences with programming Isaac rulesets, and proposes future optimizations and enhancements to the inference engine.

1 Introduction

Isaac is a project developing a rule based visual language for geometric reasoning, specifically for use in mobile robots. Some of its features include:

1. Visual representation of geometric concepts
2. Explicit representation of uncertain knowledge regarding the robot's surroundings
3. Evidential reasoning for model updating
4. Consistent handling of input, output, and inference rules[10]

In this system, all of the concepts of the language are geometric, the goal being to determine the extent to which a purely geometric formulation can be used to specify a reactive robot control environment. The model being manipulated by the ruleset is geometric; the rules themselves are activated by intersections of geometric objects in the rules with objects in the model.

In this paper, we discuss recent work on Isaac's implementation. The particular focus of the paper is on the development of a prototype inference engine interpreting Isaac rulesets, and its use in a simulator allowing testing of rulesets.

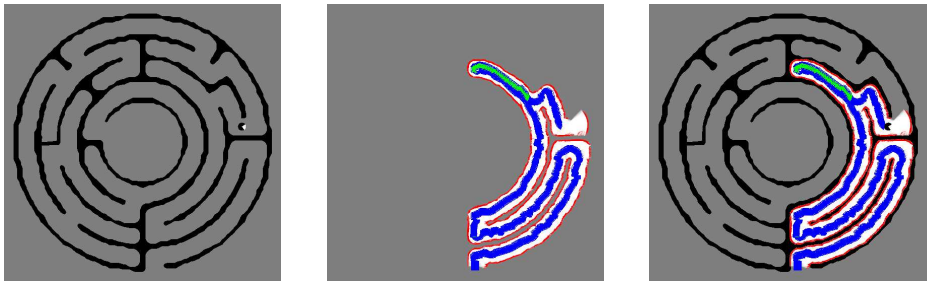
In the following two sections, we will briefly review Isaac's model and rules. Following this Introduction, section 2 will describe the inference engine proper. Section 3 describes the system's user interface. Finally, Sections 4 and 5 provide a discussion of our experiences to date with the system, and suggest avenues for future research.

1.1 Environment Model

Isaac's environment model is a set of planes representing features of interest. A separate plane is used for each type of feature; for instance, obstacles to the robot's progress or a path to be followed. Points in the planes are assigned Dempster-Shafer belief values, maintaining information as to the system's belief in the presence or absence of features at that point. This is represented in this paper as *(b, d)* where *b* is the degree of belief in the feature at that point and *d* is the degree of disbelief. The sum of *b* and *d* at a point must not exceed 1; ignorance regarding the presence or absence of a feature is represented by a sum of less than 1[12]. Model planes are processed independently; this both provides a simpler semantics for a programmer, and also avoids a combinatorial explosion in the number of subsets of planes which would need to be maintained by the belief functions.

When the planes are defined, the programmer is able to set an initial belief function for each plane. Typically, a plane that will be used to represent obstacles or other features in the arena will be given a reset belief function of *(0, 0)* (no information at all regarding presence or absence of obstacles), while planes being used to manage information regarding progress made by the robot (for instance, passageways which have been explored) will be given a reset belief function of *(0, 1)* (the system is certain that nothing has been labelled by the robot).

An example of an Isaac environment model, generated in the course of solving a maze, is shown in Figure 1 . In Figure 1(a), the arena being being explored is shown



(a) Maze and Robot

(b) Obstacles and Trace

(c) Combind Maze and Map

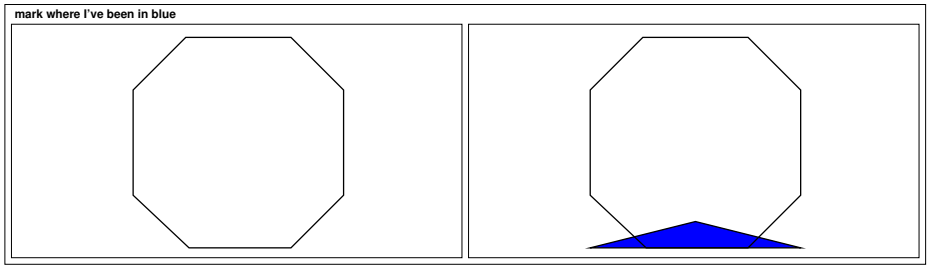
Fig. 1. Example Isaac Model

as a grey path on a black background (this map is taken from a maze in the Iveagh Gardens in Dublin, Ireland[3]). The robot is represented as a black circle with a white wedge showing the current direction of travel. Figure 1(b) shows the model which has been developed to this point in the robot's exploration. Three planes are being used for this model: in the on-line version of this paper the sides of the maze which have been discovered at this point are shown in red, while the line up the middle of the

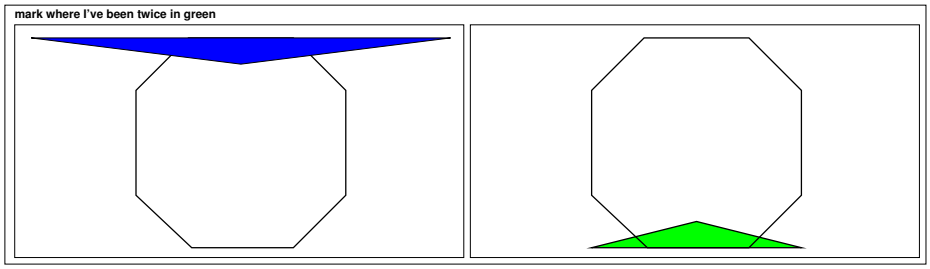
path is shown in blue[1]. In addition, the dead end which the robot has explored and back-tracked is shown in green. Saturation is being used to display confidence in the presence of obstacles with brightness being used to display confidence in their absence, as described in [9]. In this partial map, this results in the path itself being shown as white (as there is a very high confidence that there are no obstacles in the path) and the edges of the path being shown in red. The blue plane is being used to trace the progress of the robot, resulting in the line proceeding up the center of the path.

1.2 Rules

Isaac rules take the customary form of a left hand side containing predicates to be satisfied and a right hand side containing assertions to be made as a result of rule firing. In Isaac's case, the predicates take the form of geometric objects to be matched against the model, and the assertions take the form of polygons to be entered into the model. Two typical rules in Isaac are shown in Figure 2.



(a) Rule Marking First Maze Traversal



(b) Rule Marking Second Maze Traversal

Fig. 2. Isaac Rules for Path Marking

[1] For the remainder of the paper, all references to color will refer to the on-line version.

The rules are represented by two figures: on the left hand side is a geometric object to be matched in order to activate the rule, while the polygons to be inserted in the model appear on the right. These rules are used to mark paths which have been explored by the robot: the rule in Figure 2(a) has no left hand side (the empty octagon represents the robot itself; this is always shown to provide a reference for the development of the rules), so the rule is always activated. Its right hand side marks the locations where the robot has been; it is responsible for the blue line in the map shown in Figures 1(b) and 1(c). The rule in Figure 2(b) identifies a path which is being explored for the second time, and leaves a green trail to mark the second traversal of the path. This enables the use of Tremaux's Rule to solve island mazes[5].

Rules can also be used to control actuators. Figure 3 shows the rule used for following the left wall of a maze. The left hand side of this rule matches the absence of

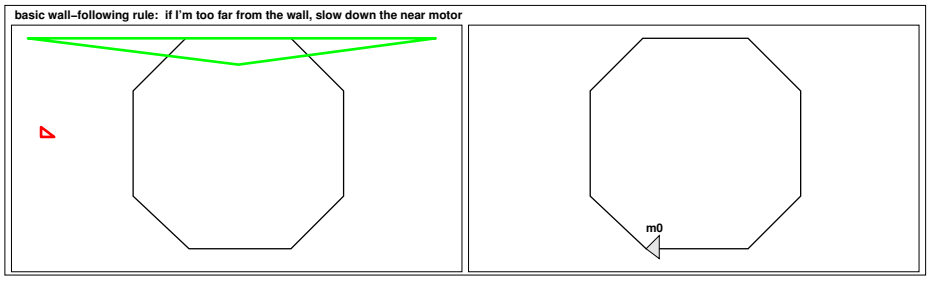


Fig. 3. Wall Following Rule

features: the empty green triangle at the front of the robot establishes that this is not the second time the path has been followed, and the empty red triangle to the left of the robot establishes that there is no obstacle at the given distance to the robot's left. If both of these triangles are matched by the lack of a green line and the lack of a maze wall respectively, the rule is activated. The grey triangle in the rule's right hand side labelled **m0** (for motor 0) controls the robot's left motor. The triangle's position and saturation control the extent to which the motor is enabled; the effect of this rule is to attempt to reverse the left motor, turning the robot to the left.

Finally, it is possible to define rules which are parameterized by sensor inputs. Figure 4 shows the rule which is used to process the forward sonar sensor. The arrows labelled **s2** are used to represent the input from sonar sensor 2, which faces directly forward. In general, an arrow can be placed arbitrarily in a rule right hand side; its origin and orientation are fixed in relation to the robot by its position in the rule, while its length is determined by the actual reading (transformed appropriately as described in [11] to reflect geometric values rather than a raw input). A point placed along (or beyond) an arrow provides a scaling factor.

The result of this rule is to insert two triangles into the model: a triangle extending from the center of the robot platform to the sensor return range marking an area where

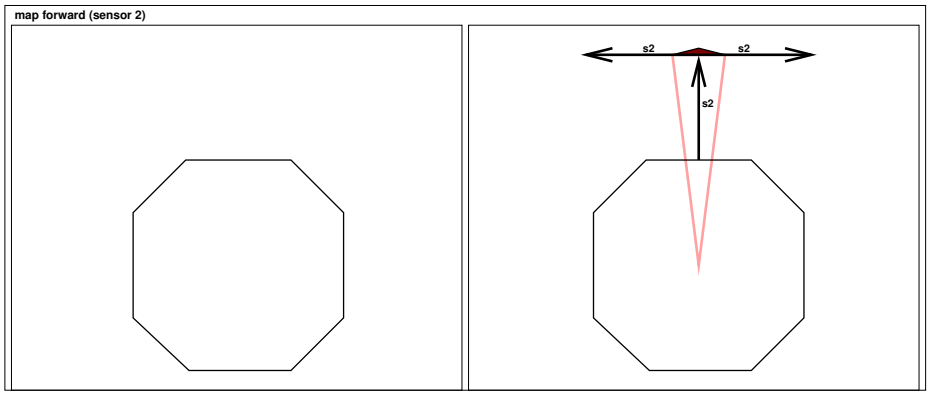


Fig. 4. Forward Sensor Rule

no obstacles are present, and a triangle at the sonar return range marking the presence of an obstacle. The triangles' widths are 25% of the sensor return value.

In addition to the model manipulation rules described here, the full Isaac system also includes navigation rules for moving the robot's position in a model[8]. In order to experiment with model manipulation, the prototype inference engine does not process these rules. Instead, the simulation maintains the correct position and orientation of the robot.

2 Inference Engine

The inference engine operates on the rules in two phases: first, the degree of activation of each rule is determined, and second, the model is updated to reflect the firing of the rules. This results in a conceptually parallel execution of the rules. In this section, we discuss the inference engine itself.

2.1 Activation

In order to determine the activation of a rule, all of its patterns are matched against their corresponding model planes. The pattern is transformed according to the position of the robot in the model, after which the model points are examined for either the presence (in the case of a filled triangle) or the absence (in the case of an empty triangle) of an object. For a filled triangle, the degree of the match is determined by the maximum model belief value in the intersection; for an empty triangle, the degree of the match is determined by the minimum model disbelief value. The intent is that for a filled triangle, we have a good match if an object is present anywhere in the triangle; for an empty triangle, we only have a good match if we have good reason to believe that there is no object present anywhere in the triangle.

A rule is activated to the extent of the worst match of all its patterns. A rule that has no left hand side is always fully activated.

2.2 Firing

After the activation of all the rules has been determined, they are all fired to the extent that they are activated, and all of the polygons on the rule's right hand side are entered into the model.

When a rule is fired, the objects making up its right hand side are transformed according to the robot's position in the environment model. Each right hand side polygon's belief values are multiplied by the rule's activation, and the model is updated using Dempster's Rule of Combination. Note that while this is described as "entering a polygon in the model," the result of the updating may be increasing confidence in an object's presence, increasing confidence in its absence, or decreasing confidence in either or both. As Dempster's Rule is commutative and associative, the order of firing does not matter.

2.3 Motors

In the motor control rule from Figure 3, the left motor (**m0**) is assigned a triangle with vertices at (-4, -8) (-3, -7) (-3, -9), with a belief of 0.05. In processing motor control rules, only the degree of belief and the y-component of the center of area of the motor's polygon(s) are considered; in this rule, the y-component is -8. There is also a default movement rule (not shown) with no left hand side (so it is always fully enabled) which assigns both motors a y-component of 8 and weight of 0.1.

A motor's speed is given by the weighted average of all of the rules that affect its speed. Consequently, if this rule were to be fired with an activation level of 1, the left motor would be set to a speed of 2.67 while the right motor would continue to have a speed of 8. The net result would be for the robot to turn toward the wall. For simulation purposes, motor speeds are adjusted so that a speed of "8" translates to a movement of one pixel in an execution cycle.

3 Simulation User Interface

A user interface has been developed controlling simulation execution and visualization, as shown in Figure 5. The interface allows the user to control speed of execution (the user is able to single-step execution, run the simulation slowly, or run it as quickly as the simulation, inference engine and user interface are capable of execution), reset the simulation to its initial state, or select a new location for the robot to begin. The user can select whether the robot and arena are visible, and independently enable or disable display of each model plane. Finally, the current speed of each motor is displayed.

The planes are assigned arbitrary hues by the ruleset designer, for use in visualizing the model as the system is in operation.

Each pixel of the model area is displayed using the visualization described in [9]. Briefly, the plane is assigned a hue by the user. Saturation and brightness are used to represent *(b, d)*, with increasing saturation being used to represent *b* and increasing saturation representing *d*. Figure 6, taken from [9], shows saturation and brightness as a function of belief and disbelief for the red plane.

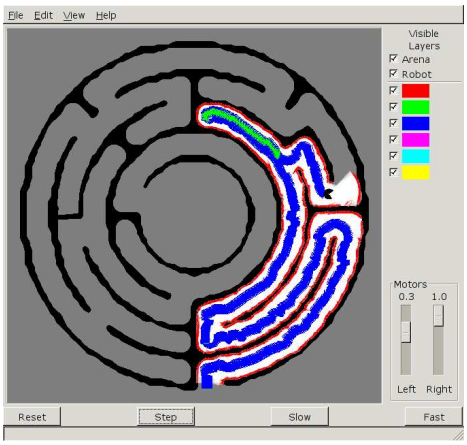


Fig. 5. Isaac user Interface

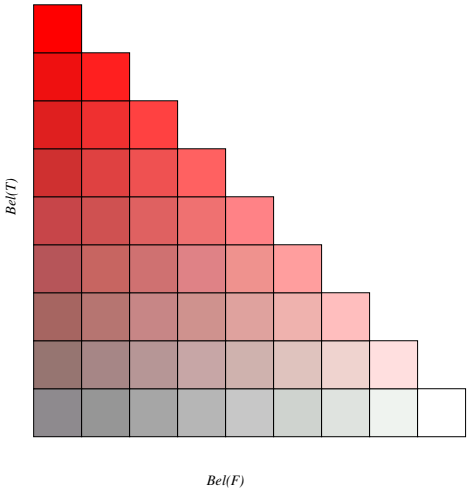


Fig. 6. Belief Visualization

When multiple layers are displayed, a crude heuristic is used to select the color at each pixel in the display. Each visible layer is examined for degree of belief and, if the reset disbelief value of the plane is non-zero (see Section 1.1), disbelief. The pixel is then assigned a color according to the following rules:

1. If the highest-belief displayed plane is greater than a threshold (the default threshold is 0.25), the pixel is visualized according to that plane.
2. Otherwise, if the arena is being displayed and has an obstacle at that pixel, the pixel is displayed as black.

3. Next, if there are any displayed planes with reset belief value other than *(0, 1)*, the visualization of the highest-disbelief such plane is used for the pixel.
4. If all else has failed, the pixel is displayed in the neutral grey used for *(0, 0)*.

A well-known pitfall in the development of visual interfaces is the use of culturally-biased or perceptually difficult colors; the difficulty color-blind users have in making use of interfaces using red and green is an example. In Isaac, the hues chosen for color planes are not semantically meaningful. Planes are assigned hues by the ruleset programmer, however, the hues can be changed at will by the user of the simulation by selecting the color palette buttons on the right side of the interface window. This doesn't prevent the use of inappropriate hues (imagine a user selecting the same hue for two planes), but does at least provide a mechanism for the user to make better choices.

4 Discussion

Experimentation with a variety of arenas and rule sets has led to a number of observations regarding Isaac's evidential geometric reasoning.

First, it has proven possible to implement a number of standard reactive robotic tasks, such as obstacle avoidance, mapping, and maze solving. The use of levels of belief in the objects to be inserted, and especially in the motor controls, have led to a programming methodology which is quite similar to Brooks's subsumption architecture [1]. The common methodology for creating a ruleset has been to begin by creating a basic rule for robot movement, and then create new rules with greater weights for responses to particular situations. The weights are then adjusted in response to the simulated robot's performance in assorted arenas.

Several intuitions regarding the development of rulesets have proven to be false. It was anticipated, for instance, that it would be most useful for objects appearing in rule left hand sides to be large, so it would be possible to identify objects in a large area. This approach led to rulesets that did more to expose "holes" in the robot's sensor coverage than to solve their intended problems (the rule shown in Figures 3, with its very small area of non-obstacle in the left hand side, is an example of a rule developed following the development of this insight). For rules not based on maps directly derived from sensors the result has been exactly opposite, with very large regions in the rule left hand sides picking up marks left behind (as seen in Figures 2 and 3).

5 Future Work

The development of this inference engine, and our exprience with it, has led to a number of possibilities for future work.

5.1 Rulesets

At present, the inference engine only handles model manipulation rules, not navigation rules (the simulation simply maintains the robots position, and the correct position is used in rule evaluation). This has proved quite useful in exploring the language itself, but these rules need to be added both for a more realistic simulation environment and to permit the system's use in an actual mobile robot.

5.2 Data Structures and Algorithms

The data structures and algorithms being used in the prototype are very simple, due to the interest in having the system operational in as short a time as possible: the model is implemented with an array of pixels for each plane, and updating objects in the model is performed by updating each pixel in the object. Also, every rule is evaluated on every processing cycle.

A two dimensional geometric library using simple polygons directly as the geometric primitives is currently being implemented, with the intent that it will replace the array-based model currently in use. It is anticipated that this will exploit spatial coherence to significantly reduce both space and time requirements.

Algorithmic improvements are of greater interest. There are well-known algorithms, notably Rete[4] and TREAT[6], which act to evaluate rulesets more efficiently than the naive approach described here. Rete has been extended to fuzzy production systems [7]. We expect to exploit the new model representation to extend the the fuzzy Rete algorithm to a geometric fuzzy Rete.

Another avenue for algorithmic improvements is parallelism. Isaac's rule processing semantics were designed to be conceptually parallel; the activation level of all rules is calculated, and then all rules with a non-zero activation level are applied using only commutative and associative operators. Consequently, a parallel implementation of the inference engine should be feasible. At the very least, it is possible to separate the processing by planes in a shared-memory environment.

5.3 Visualizations

The visualization which was developed for this prototype has proved quite useful in observing the behavior of the robot and the rules in exploring environments. At the same time, experience has shown some deficiencies which we are working to address.

First, while the heuristic being used to visualize the arena, robot, and model is reasonably successful (especially with a user interface that enables display of the individual planes easily), there is no sense in which multiple planes with relevant information can displayed at a single pixel. We are investigating blending functions which might better display more of the information visible at a given location in the model.

Another area in which we hope to augment the current global visualization is by adding a local, robot-centric visualization to better display the rules currently under consideration. A common debugging technique with the current visualization is to define a small triangle in an otherwise-unused model plane in the right hand side of a rule, so the degree of activation of the rule is left as a trail behind the robot. While this has proved useful, it would also be very helpful to display a large view of the robot itself in context of the model, and showing the polygons on the left hand sides of some of the rules with a saturation showing their degree of activation.

5.4 Integration

Work has previously been done in the design and prototyping of the user interface for a visual editor for Isaac, but this has not progressed to the point of the visual editor being

able to actually create ruleset files. Instead, the figures showing visual representations of the rules were generated with the xfig drawing tool, and translated by hand into the textual representation. This has actually been quite instructive, as attempts to shortcut the process by directly writing textual rules has shown a surprising level of difficulty in imagining the geometry of the objects in the rules. Further development of this editor, so creation of rulesets will be directly possible, is another area in which we are proceeding. It is interesting to speculate on the possible results of combining the robot-centric visualization described in Section 5.3 with the rule editor, so it could become possible to create new rules based on a situation present in the model. Another approach to this combination, focussing on a direct manipulation of the robot model, can be found in [2].

Acknowledgements: This work was partially funded by National Science Foundation MII grants EIA-9810732 and EIA-0220590.

References

1. Rodney A. Brooks. A robost layered control system for a mobile robot. *IEEE Journal of Robotics and Automation*, RA-2(1):14–23, March 1986.
2. Phil Cox and Trevor Smedley. Visual programming for robot control. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 217–224. IEEE, IEEE, 1998.
3. Jo Edkins. Jo edkins's maze page. www.gwydir.demon.co.uk/jo/maze/.
4. C.L. Forgy. Rete: a fast algorithm for the many pattern/many object pattern match problem. *Artificial Intelligence*, 19(1):17–37, September 1982.
5. Edouard Lucas. *Recreations mathematiques*, volume 1. 1882.
6. D. P. Miranker. Treat: A better match algorithm for ai production system matching. In *Proceedings of Sixth National Conference on Artificial Intelligence*, pages 42–47, 1987.
7. K. Pan, G.N. DeSouza, and A.C. Kak. Fuzzyshell: A large-scale expert system shell using fuzzy logic for uncertainty reasoning. *IEEE Trans. Fuzzy Syst.*, 6:563–581, 1998.
8. Joseph J. Pfeiffer, Jr. A language for geometric reasoning in mobile robots. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 164–171. IEEE, IEEE, September 1999. Tokyo Japan.
9. Joseph J. Pfeiffer, Jr. Using brightness and saturation to visualize belief and uncertainty. In Mary Hegarty, Bernd Meyer, and N. Hari Narayanan, editors, *Diagrammatic Representation and Inference, Second International Conference, Diagrams 2002*, pages 279–289, April 2002.
10. Joseph J. Pfeiffer, Jr., Rick L. Vinyard, Jr., and Bernardo Margolis. A common framework for input, processing, and output in a rule-based visual language. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 217–224, Seattle, Washington, USA, September 2000.
11. Joseph J. Pfeiffer, Jr., Rick L. Vinyard, Jr., and Bernardo Margolis. A common framework for input, processing, and output in a rule-based visual language. In *Proceedings of the IEEE Symposium on Visual Languages*, pages 217–224. IEEE, IEEE, September 2000. Seattle USA.
12. Glenn Shafer. *A Mathematical Theory of Evidence*. Princeton University Press, 1976.