8-17-2005

Self-Adjusting Finite State Machines: an approach to Real-Time Autonomous Behavior in Robots

Scott Schwartz
Trinity University

Self-Adjusting Finite State Machines: An approach to Real-Time Autonomous Behavior in Robots

Scott Schwartz
Department of Computer Science
Trinity University
One Trinity Place
San Antonio, Texas
78212-7200. USA

Faculty Advisor: Dr. Maurice Eggen

Abstract

In the Robotics industry, it is a frequent requirement that robots operate in real-time. The usual approach to this issue involves creating robots driven entirely by direct environmental input rather than complicated planning and decision-making AI. This approach means that the current state of the robot in relation to its environment exclusively determines the actions of the robot. In the simplest terms, this approach creates a Finite State Machine (FSM). Clearly, a standard FSM is completely pre-deterministic upon its creation. This is a drawback which immediately disallows the robot to cope with dynamic environments in an autonomous manner. This research suggests a solution to this problem, while still maintaining real-time performance of the FSM structure, through the development of a Self-Adjusting FSM (SA-FSM). A SA-FSM is a FSM with an additional module which adds, removes, and adjusts specific states of its FSM structure. By adjusting its FSM the SA-FSM will have the basis for autonomous attributes. It will be capable of coping with drastic changes in its environment by making necessary fundamental adjustments to its behavior. Through this mechanism, the process of learning can be implemented. In this regard, only the inherent learning/inference algorithms the SA-FSM employs to adjust its FSM determine the complexity of the behavior produced by a SA-FSM based robot.

Keywords: Robots, Autonomous Robots, AI, Real-Time, FSM.

Self-Adjusting Finite State Machines: An approach to Real-Time Autonomous Behavior in Robots

by

Scott Schwartz

Acknowledgements

First and Foremost, I would like to thank my parents, Mom and Dad, for their continual support in my academic career. Their interest and intervention is this as well as all other areas of my life brings warm feelings. On this topic especially, their attention has been very comforting.

I would like to thank Dr. Gerald Pitts and Dr. Maurice Eggen for initially providing the opportunity for this research. I would like to further thank Dr. Maurice Eggen for always teaching excellent classes, and for being my trusted advisor throughout this research. In general, I would like to thank each member of the Computer Science faculty who I have been involved in during my academic career here at Trinity. The same is goes for my professors in the Mathematics department.

Also, a strong recognition of thanks must be extended to Dr. Diane Saphire, my statistics professor. Her teaching has greatly influenced my entire life, and thus, influenced this work.

Additionally, I have enjoyed the company of many fellow students while here at Trinity. Thank you all for the fun times.

Finally, I would like to thank pretty Sarah Maspero for being so pretty, and being an excellent distraction from this work.

Table of Contents

Chapter 1

Introduction

Robot. The word stimulates a variety of captivating thoughts. When I see this word I think about the future, about advancement, and about better times. I think about the benefits society would gain from conveniences that could be achieved through robots. I think about all the difficult tasks that humans cannot perform on their own which could be accomplished with the help of robots. And I think about dangerous situations which would no longer require direct human intervention by instead using robots. Then, I begin to imagine creating a conscious, intelligent robot. Or, at least maybe one that just seems to display such characteristics. From this point the imagination can extend these reflections into the most meaningful and intriguing realms ever considered by human minds. But this reflection on the word *robot* is almost exclusively influenced by the cartoons I watched on TV as a child and the movies I see from Hollywood today. Is any of this real? Is any of this even possible? If it is, where is it? Where are all the robots now? What really is happening with robots today?

1.1 Robots Today

An official summary of the state of the robotics industry today can be found in the *2004 World Robotics* Survey [4]. This report was issued by the United Nations Economic Commission for Europe (UNECE) in conjunction with the International Federation of Robotics (IFR) [4] [5]. There is a wide range of types of robots in use today. From the current state of the robotics community, it would appear that the world is very close to becoming dominated by robot usage. This has not yet occurred, but the majority of components for such a shift all appear to be in place. What this will mean on the personal, cultural, and global level is a very interesting question. The effect of such a move on society is not merely a black and white issue, but will have a number of varied affects. This discussion is not the topic that is presented here, although by entering into the robot domain it would seem that this issue should be given significant consideration for its ethical implications. Nonetheless, we shall continue.

1.1.1 Industrial Robots

The first group of these robots is the industrial robot group. There are currently just less than one million industrial robots in use today. Sales in this market are currently increasing at a yearly rate of 7%. In keeping with the group name, industrial robots are used primarily in industrial factory applications. The main two areas for their use are welding applications and assembly line applications. These areas take up 25 % and 33% of the market, respectively. The remaining industrial robots are used for other specialized purposes. Currently, these robots are especially prevalent in the automotive and electronics industries. Industrial robots are expected to play a major role in food production and packaging industries in the near future.

1.1.2 Service Robots

A second classification is service oriented robots. These are robots that operate semi or fully autonomously to perform services to humans in a non-industrial setting. Service robots are divided into two classes.

The first are robots for professional use. There are currently only 21,000 professional use robots. But by 2007, 54,000 new units are expected to be in use! Underwater systems make up 23% of this category. Cleaning robots make up 16%. Laboratory robots make up 15%. Demolition and construction robots make up14%. Medical robotics makes up 12%. Mobile robot platforms for general use make up 9%. Defense, rescue, and security applications make up 5%. And field industry robotics like forestry and milking machines make up 4%. Optimistic projections show that new areas with potential for the most growth in this category involve humanoid robots and public relations robots.

The other category of service oriented robots is those for personal and private use. There are currently 610,000 domestic robots and 700,000 entertainment robots today. This area of robotics is booming with a projected 4.1 million new domestic robots and 2.5 million new entertainment robots by 2007! The main types of robots comprising this area of robots are vacuum cleaning and lawn-mowing robots, as well as entertainment and leisure robots.

1.2 Robots Today

In the category of service oriented robots for professional use, there is currently a growing interest in autonomous mobile robot research. *An autonomous mobile robot is a robot that independently navigates and operates in its environment in order to perform its tasks* [3]. It may also need to learn and adapt to its environment [3]. Many research institutions and the majority of universities have projects devoted to autonomous mobile robot projects. The underwater robots, laboratory robots, and mobile robots that are already in operation today are just the meager beginnings of what is expected from autonomous mobile robots. The DARPA Grand Challenge showcases a race of unmanned vehicles [6]. There is a RoboCup which showcases robot soccer teams playing against each other [7]. There is even an annual International Fire Fighting Robot competition [8], as well as a numerous variety of other autonomous mobile robot competitions.

Just as the extensive range of applications of the competitions suggests, autonomous mobile robots will have an extensive range of application and impact and will be useful for many purposes. They can be designed and built for a variety of objectives in number of different environments [3]. Already, there are applications for autonomous mobile robots in a variety of environments set in the air, underwater, and on land. Autonomous mobile robots can be used to explore, investigate, and search areas where, for whatever reasons, humans cannot go [2]. They can perform tasks that humans are unable to perform such as the interaction, collection, and transportation dealing with dangerous materials or environments [1]. On a less glamorous level, autonomous robots can be used for more mundane tasks that are too dirty or too dull, such as cleaning, guidance, inspection, and surveillance [1]. The possibility for their use is limited only by our own conceptions about their goals.

1.3 Focus

While all of the goals of different types of autonomous mobile robots are not the same, there is certainly a unifying similarity in the overall field. To facilitate the research however, autonomous mobile robots will henceforth be approached through a specific instance in this field. This instance will be an autonomous mobile robot for use in planetary exploration. The current benchmark in this area is the twin robots Spirit and Endurance which are currently operating under the Mars Robot Exploration Mission [9].

However, these robots are not completely autonomous and must receive instructions from human controllers. Additionally, they do not perform as efficiently and effectively as a truly autonomous entity. There is still progress to be made in this area [10]. The overriding goal is to contribute to the advancement of this type of autonomous mobile robot. As an added benefit, progress in this area is sure to positively influence other areas of the autonomous mobile robotics field due to the related nature of this whole discipline. Nonetheless, from now when using the term robot, we are referring to a mobile robot of an exploratory nature.

Chapter 2

Mobile Robot Mechanics

This section will review the engineered equipment that is available to be used in constructing a mobile robot. Specifically this section focuses on components involved in the mechanical movement of robots as well as components which provide a way to perceive and represent the environment outside of the robot. These are only mechanical pieces of the robot. They make up the ability to create a mobile robot. At this stage, the robot will have no control over itself, but can still be operated in a useful manner through human direction. But also at this state, the robot can be given the necessary additional components which can provide that it will be autonomous. These mechanical components alone are only tools for helping reach this goal of developing an autonomous mobile robot. They present several problems of their own which must be studied and addressed which do not relate to autonomous mobile behavior. But they are to be used towards solving this objective, as they are necessary components for its completion. It is worthwhile to understand or at least become familiar with the general nature of these components so that this information can be taken into account in order to facilitate the development process instead of hinder it.

2.1 Focus

The whole premise of autonomous mobile robots is to allow an intelligent entity to move about its environment in order to accomplish a series of assigned tasks [3]. How can the robot to traverse its environment? There are a number of options for the question of movement. The robot may possibly require some combination of various capabilities. A tool which the robot interacts and makes changes to its environment is called an actuator. Bringing together the correct combination of actuator capabilities will require a carefully designed robot. The robot plan will be directly dependent upon what the robot will need do. The robot will need to be built for its environment [1]. There are a number of logistic issues which must be answered in order to bring all the required forms of motion together. Issues such as overall weight, actual space available on the robot, and energy distribution must all be addressed [1]. It is clear that the design of the robot is

of the utmost importance [1]. However, the issue of energy is at least of equal, if not greater, practical importance in the use of mobile robots. Energy is a precious commodity in mobile robots. Creating lasting and renewable energy source for a mobile robot is by no means solved, and is a key focus of research in robotics today [2].

2.1.1 Ground Movement

In the case of an exploration robot, it is clear that ground movement over a diverse set of terrains will be required. The most prevalent approach to movement is to use a wheeled robot. A wheeled robot is simple to construct and is efficient with respect to weight capacity as well as energy use [1]. However, wheeled movement can be poor over uneven terrain, so if this approach is used special care must be given to the design and configuration of system in order to offset this issue [1]. There are many options to be addressed in this type of a system. What strength shocks are required? How many tires should be used, and in what configuration should they be arranged? What type of tire should be used? An alternative which addresses the problem of movement over difficult terrain is to use tread instead of tires. But this approach is generally not very energy efficient because of the nature of treaded movement [1]. There is also a set of questions for treaded movement similar to those for wheeled movement. A very divergent approach to the above options is to use legged motion. This approach generally involves biomimetics, which is the use of living creatures as inspiration for machines [2]. This approach has already been verified through the organic creatures in our world. But, designing these systems is much more complex than the other approaches, and requires many more mechanical parts to be useful [1].

2.1.2 Other Forms of Movement and Interaction

In addition to land movement, the robot could also require other forms of movement. It may be required to take to the sky in certain situations. Or, perhaps it will need to travel on or in an ocean-like environment. Ariel robots are usually mimic helicopters or blimps, and require a lot of design and planning, as well as a large amount of energy. Water related robots resembling boats and submarines have already had some success to date. The issues here will be incorporating these components into the design of the robot. Besides general movement abilities, there will of course be other ways the robot will be required to interact with its environment to accomplish any

specialized tasks it may need to. It may need to take pictures. It may need to grab, hold, and handle objects in the environment. It may need to protect itself. Adding these additional layers of actuator functionality again raises logistic issues for the robot, as well as complexity in general. Just as the nature of each of these forms of movement is very different, so the responses they receive in the environment will be. This must be taken into consideration. Each form of actuation must be considered on its own as well as in conjunction with the other forms of actuation.

2.2 Other Forms of Movement and Interaction

Now that a number of possible ways a mobile robot can operate in an environment have been briefly covered, the next logical step in the discussion is how the autonomous mobile robot is to sense its environment in order to act in it. It has been given the means to accomplish its tasks, but without being able to sense it will obviously be unable to complete its goals. Humans have an effective sensors array, using vision as the primary sense [1]. When humans build robots, they often design them in their own image [2]. But this is not necessarily the only way, or best way, to sense the environment. The rest of nature often does their sensing of the environment differently [1]. Dogs use their sense of smell to an elevated degree compared to humans. Bats use an amazing sound based system to view their world. Cockroaches use a sophisticated feeling system to sense their near and distant surroundings. Unfortunately, robots do not currently have access to the organic sensing systems that the creatures in our world use. Until, that happens, we must settle for mechanical attempts and sensing. Some of these are remarkably advanced and effective, while others have not yet quite been mastered. What methods are available for the type of robot application in question for sensing the environment? Sensors for Mobile Robots: Theory and Application by H. R. Everett contains a detailed introduction into this question [13]. The following is based off Everett's presentation of the topic.

2.2.1 Position Location Systems

Usually mobile robots will be designed to take into account dead reckoning. This is the process of locating where the robot currently is in relation to where it has been. A simple technique for of dead reckoning is odometry. This is where the motion actions are monitored and used to calculate how the robot has moved, and thus the location of

the robot. Odometry sensors include, brush encoders, potentiometers, synchros, resolvers, optical encoders, magnetic encoders, inductive encoders, and capacitive encoders. The problem with all of this is that it is based entirely off of the movement generated by the mobile robot. There is no way to monitor externally motivated changes to the robots location such as slipping. This problem will require a whole new set of sensors. Fortunately, there are different approaches to dead reckoning. These include Doppler and internal navigation. The idea here is to compare the robots position directly from its actions relative to the environment, instead of through only its movements. Doppler monitors the direct environment to determine how the robot is traveling. Internal Navigation involves sensing the accelerations of the robot, and directly computing its resulting location.

Locating the specific location of the mobile robot can be done in other ways. There are ground-based and satellite-based radio frequency position location systems. There are also ultrasonic and optical position-location systems. These are forms of GPS (Global Positioning System). This methodology locates the position of the robot in its environment through satellite or other similar means by use of radio waves. Generally, a chip is put into the mobile agent and is then tracked through one of the above methods. This is a new technology, and there is currently much ongoing research in this area. So far, this appears to be a very feasible and effective system.

2.2.2 Compasses and Gyroscopes

In addition to the mobile robot knowing its location, it would probably also like to know certain attributes of its orientation. This can be done with compasses and gyroscopes. Compasses of course give a direction or a heading. There are a remarkable number of choices in this sensor group. There are *mechanical magnetic, fluxgate, magnetoinductive magnetometers, hall-effect, magnetoresistive,* and *magnetoelastic compasses*. Gyroscopes on the other hand can do a little more than compasses. They give measurements such as pitch and tilt. There are two categories of gyroscopes, *mechanical* and *optical*. In the first category are *space-stable gyroscopes, gyrocompasses,* and *rate gyros.* In the second category are *active ring-laser, passive ring resonator, open-loop interferometric fiber-optic, closed-loop interferometric fiber-optic,* and *resonant fiber-optic gyros*.

These two forms of input, compass and gyroscope, both give mobile robots a better interpretation of their environment, and how they are situated in it. These sensors

are just another step in making sure the robot is more spatially aware. This type of information will be critical to the well being of the robot.

2.2.3 Proximity and Touch

These sensors will be critical to the robot's success. It would probably be a good idea to equip the mobile robot with collision detection because it will be a bad idea to allow the robot to smash into dangerous obstacles. It needs to be aware of obstacles and avoid them. Further, if the robot is hit, it needs to realize that and react accordingly. Proximity sensors alert the robot to the presence of an obstacle. Some give a general range for the obstacle, and some do not. This is crude, but useful. There are several categories of proximity sensors. There are *magnetic, ultrasonic, optical, inductive, microwave,* and *capacitive* sensors. These sensors will allow the robot to be alerted when an object is within a certain range of the sensor. Slightly more simplistic than the proximity sensors is a tactile sensor, which can alert the robot when it has been touched. Technology in tactile sensors includes *contact closure, magnetic, piezoelectric, capacitive, photoelectric, magnetoresistive, piezoresistive,* and *ultrasonic* sensors. Basically, these sensors are configured as tactile feelers, tactile bumpers, or distributed force arrays. Proximity sensor technology is improving, so touch sensors are not quite as useful as before when compared to proximity sensors, but there still can be uses for them. And, in some circumstances, they will be preferred over proximity sensors. In both types of sensors, the configuration that the sensors are applied onto the robot will be determined as part of the design, and are based off of the needs and requirements for the robot.

2.2.4 Ranging

The next step beyond simply being aware of obstacles through proximity and tactile sensors is to be able to quantify the position of the locations of objects in the robot's environment. This will allow intelligent movement and interaction of the autonomous mobile robot with its environment. As in the last case of sensors, the configuration of these types of sensors must be suited to meet the needs of the robot. There are several techniques in this area, and each of these has several variants. *Triangulation* is one such method whose variations include *stereo disparity, single-point active triangulation, structured light, known target size,* and *optical flow* approaches.

These approaches generally involve the comparison of two angles focused on an object in a vision related methodology. Many of the techniques in this area can generally be implemented as either *active* or *passive* sensors, with the choice depending on the system. The first term simply means the sensors will radiate energy into the environment they survey, such as the common approaches to radar, sonar, and lidar do. The second term is the opposite of the first. Another method is *time of flight* which is usually implemented in ultrasonic or laser form. Distance is measured by releasing a pulse of energy and measuring the return time. Some methods which are used for distance calculation but are not yet widely used for different reasons are i*nterferometry, range from focus,* and *return signal intensity.* These approaches may become very useful soon. An advance even beyond determining the distance of an object is to detect any acceleration or velocity that the object possessed. This is possible through continuous wave approaches such as *phase-shift measurement* and *frequency modulation*. In this approach a wave is sent out and the part of the wave that is reflected back can be used to interpret active attributes such as velocity and acceleration.

2.2.5 Environmental Occurrence Sensors

Any given autonomous mobile robot will need to be suited for the tasks it is built to accomplish. Each of these robots will need to be equipped with the correct sensor array so as to be able to cope with its tasks. There are many other sensors which are used for specific tasks that are available to help a specific task robot to accomplish its goals. Sound sensors are such a sensor. While acoustics are already used for ranging, a passive use of sound, as well as other forms of active use can also be beneficial in autonomous mobile robots just as it is in humans. As with sound, electromagnetic waves are already applied for visual sensor applications. Just like sound, the electromagnetic spectrum could be used for communication. The higher and lower frequencies which are not used for visual applications could still be employed. Sensors detecting temperature levels would likely be important, especially for the well being of the robot. As with temperature, other forms of radiation sensors might be necessary. Detecting vibrations could likely be important for the well being of the robot. Additional presence and motion detectors could also be very useful. Odor sensors might serve some special purpose in a certain robot. Other important and useful sensors would be status indicator sensors which monitor the robot's components themselves as opposed to monitoring events that are external to the robot. The point is that are a very wide

number of sensors applications that would be useful for certain mobile robots. These sensors are available to be employed in the use of autonomous mobile robots. Whatever the application the robot is to be used for, there is likely a sensor which will help with that application [1].

2.2.6 Sensor Considerations

So, as far as sensors go, it appears that whatever the application, we are likely to be able to construct a sensor which will adequately convey the situation to the autonomous mobile robot. But is this really the case? Are all these sensors as reliable as we would hope they should be? The simple answer is that sensors are usually not 100% reliable. They can be completely wrong, but usually the problem is simply that their measurements are not precise. The sensors are full of noise. The causes are different in different sensors, and they are often unavoidable. This is an issue in mobile robots that will have to be taken into consideration and adjusted for.

So while it at first seemed that sensing the environment would be easy with all the technology that is available to us, it may not be so easy in actuality. Further complicating the situation is the fact that it is probable that the robot will use a vast array of different sensor types. As the robot becomes more and more sophisticated, the number of sensors required to maintain the higher level of sophistication also must increase. When considering sensors, it's important to note that just as these different sensors are very distinct and so their input cycles will also be very distinct. The different sensors will have different processing time. The inputs follow different cycles. This is an issue which must be addressed, and no doubt will add additional complexity to the functioning of the robot.

Chapter 3

Autonomous Control

The preceding summaries gave a general survey of the existing mechanical components available for autonomous mobile robots. In summary, the possibilities these afford are quite extensive. The same can almost be said about the ways in which the environment can be sensed by the robot. The facts about the mechanical components by themselves are relatively unimportant in and of themselves. They are just good to be familiar with in order to give us a knowledge and understanding about what we can expect to be able to do. But how can we make the robot do whatever it is that we want it to do? How do we control all these pieces so as to actually generate an autonomous mobile robot? This question is largely a question for the field of *Artificial Intelligence* (AI). And this is the question that is being explored here. We will be interested in how we can use the potential capabilities machines give us in order to make a robot that behaves intelligently in order to accomplish certain goals. There has already been extensive research conducted in regard to this question. There are currently two major paradigms for creating the intelligence required for a robot to behave autonomously [1] [3].

3.1 Traditional Approach

The first approach is sometimes now referred to as the *traditional* approach because this, simply enough, has been the approach that is traditionally used [1] [3]. The first attempts to create an intelligent program which could operate a mobile robot in an autonomous manner involved this paradigm. In this methodology, the current state of the environment is perceived through the sensors, and then it is modeled and a plan is of action for how the robot should best proceed is developed [1] [3]. This is a sense, then think, then act approach [3]. This cycle is best represented and understood in terms of the following steps [1] [3]. The sensors record the input. Any computation required to use the sensor input is then done, and last minute errors searched for. Once the sensors input is in a usable form, it is all mixed together to represent the current state of the environment for the robot [1] [3]. This model representation of the environment is then used by the robot in order to decide how it should proceed [1] [3]. Once the plan is

developed, the robot will execute the plan. The cycle will then repeat.

This approach is also sometimes called the *functional* approach because of the nature of its operation. It receives an input, and returns an output. It is also sometimes called the *symbolic* approach because it builds up a symbol base to represent its environment and its own actions as they interact with the environment in order to produce a plan of action. Additionally, it has been termed the *deliberative* approach, because of the emphasis on the planning that goes into performing actions. There are a number of problems with this approach when it is considered on its own. Because of these drawbacks, this methodology has met with only modest success thus far in applications related the robots discussed here. The first problem is that this type of system is not very robust. It is an algorithm requiring correct execution from each of its parts [3]. Also, this approach requires an intensive computational process which may create a bottleneck which adversely affects the environmental sampling rate [1] [3]. This same problem may also slow down the reaction time of the mobile robot [1] [3]. This could be particularly disastrous in our application. During the time the robot is unable to respond the environment while it processes some information about the environment it is utterly self defenseless and useless. But a deliberative approach to problem solving would seem to have a place somewhere [1] [3]. Planning does have some merit as a concept [3]. So, this approach is still an important player in many areas of AI research. It just cannot be the only approach because of the inherent problems it entails.

3.2 Behavioral Approach

Fortunately, there is another alternative. The second approach for controlling autonomous mobile robots is called the *behavioral* approach. This is based on the *subsumption* architecture developed by Rodney Brooks [11]. Brooks has become very revered and respected for this idea. At its onset, this approach was quite novel compared to the current practices of the time. Because of this, it did not gain immediate acceptance. This is definitely no longer the case now though, as Brooks is one of the most revered figures in robotics control, and this approach is now almost universally accepted to be (at the very least) a key component for making autonomous robots. As you will see, this methodology is a very straightforward and intuitive attempt to tackle the question of autonomous behavior. The fact that this very simple approach based upon only a straightforward observation has become so useful in this field is very promising for future research.

In this paradigm, the environment is used as its own best model, and the robot simply reacts to occurrences in the environment [11]. For this reason this approach is sometimes called the *sub-symbolic* approach and *reactive* approach. The behaviors all run in parallel, simply waiting to be triggered [11]. Once a behavior is triggered, its commands are fired and the resulting action adjusts the robot's position relative to the environment, possibly triggering additional behaviors. The individual behaviors are not meant to be complex, but by combining and layering a number of reactive behaviors, an advanced and complex intelligence begins to emerge [11]. So the instructions for the robot's actions and behavior are directly from the environment around the robot [11]. This seems to make enormous sense. If the robot can be equipped with the correct sensor array in order to adequately sense the world with respect to its own well-being and its goals, it can then perform its actions in the world as the world allows.

This approach so far appears to hold great promise. Everyone who now works in this area of research is interested in some part of this methodology. It is simple, not requiring megalithic hierarchical programming achievement [3]. And rather works on simple behaviors. Because of this, it is easy to extend. Adding new behaviors does not require in massive adjustments to the current system [3]. It supports multiple parallel goals through independent individual behaviors [3]. It is very robust [3]. If one component behavior has been lost, this fact need not effect the execution of the other behaviors. There is no computation to create a time bottleneck on the system. The robot simply receives input which it is designed to react to. This approach has already met with impressive success because of the advantages that it offers. Insect-like intelligence has already been demonstrated through the use of this methodology [11]. However, there is on shortcoming in this approach. The disadvantage is that this approach does not seems to provide an easy way to allow developing and reasoning about a plan, as the *deliberative* approach does [3].

3.3 Hybrid Approach

We can not likely deny that being able to plan actions in an environment is almost a requirement for intelligence. We ourselves would say that we reason and make plans concerning ourselves and our environment. It would appear then that at least some form of reasoning and planning in an environment will be necessary for autonomous mobile robots. The *behavior* approach, which seems so promising for so many reasons, will likely fall short in this regard. We will use as much of that approach as possible in order

to allow for continuous interaction with the environment not hindered by computational interference, but it seems we will also need to employ the *traditional* approach to some extent. An autonomous mobile robot of the nature we are interested will need to continuously react in its environment as well as make overriding plans in order to accomplish it goals. The approach used in this project will be a *hybrid deliberative/reactive* paradigm. So, there will be a place for both approaches, but the ways in which the different approaches can be used may be very selective. While the *hybrid deliberative/reactive* paradigm was not discussed as its own category, approaching mobile robot control through this paradigm is becoming standard for high level applications. Continuing in this new convention seems a wise choice. While the *reactive* approach is extremely appealing for all the reasons that have been given, we need not limit our possibilities by confining ourselves to just that approach.

3.4 Machine Learning

Coupled with the process of controlling an autonomous mobile robot, is the concept of learning. In several situations it is plausible in that learning would be irrelevant or unimportant. But for an autonomous mobile robot exploring a foreign planet, it is hard to see how to proceed without some form of learning. If we didn't need to learn anything about the place that was being explored, why would we send a robot there in the first place? Learning is an interdisciplinary field of very intensive study because of its intrigue and potential importance. AI research is very involved in this field for these reasons as well as the application to computer intelligences. When a mobile robot is initially designed, many of the attributes and behaviors of the robot controller can be built in by the designers of the robot. They have an idea of the types of things the robot will be required to do and can plan accordingly. But, in robots like the exploration robot there will certainly be things that the designers cannot plan for in the control of the robot. There may be some things which occur that the robot is not already specifically designed to deal with. How then is the robot to cope with these situations? The answer is that the robot will need to learn to handle the situation. The robot will need to learn new actions to perform in these unexpected occasions. Aside from simply being an almost indispensable tool for an autonomous mobile robot, the topic of learning is a very interesting and stimulating topic to explore. Autonomous mobile robots provide an excellent opportunity for exploring this topic.

There are a number of ways to consider the topic of leaning in mobile robots.

Common approaches in mobile robots include *reinforcement learning, probabilistic reasoning,* and *connectionism or artificial neural networks* [3]. These are all methods of *machine learning*. This means that the learning is internal to the machine. This separates the robot from the outside world because the robot has only a perceived notion of the world. Machine learning is what goes on inside of the robot's conception of the world in order to acquire a new skill or a new knowledge [3]. Learning could be as simple as building a mapping of the environment. Or, it could be learning which choice of several choices is optimal under a certain setting. It could also be learning how certain objects or agents in an environment effect and make changes to the environment, and then how it should best respond to these objects and agents. There will be a number of ways to do each of these tasks. We always want to choose the best way. To do this, we will use aspects from the different current methods to machine learning where they are especially useful. We will especially rely on a statistical approach to learning. This will give us a sound theoretically based platform for building a robot which can learn and infer about its environment. Generally, it seems that learning is done through repetition. After touching a hot surface enough times, one generally does not touch it again. All learning could conceptually be considered from a statistical point of view. There are of course complications with this idea. For instance, how does teaching fit into this thought? Would it be that some of the repetitions are weighted higher than others? Questions similar to these will be explored while this method is used. But the notion of statistical learning seems like a valid base to start with when considering learning.

3.5 More on Statistics

This author believes that the application of probabilistic and statistical approaches to learning in AI. This is turn will hold promise for decision making in AI. The issues of decision making and learning are key in AI. There are many approaches to these topics, but one which holds great potential benefit can be found in applying stochastic and statistical methods to learning and decision making. To date this has not been given adequate attention in complex AI decision making and learning. This has been due in part to the cost of computations required for statistical methods being too high and in part it has simply been neglected so its application in this area is unexplored. A large part of the reason successful autonomous AI has not yet been developed is that all efforts to date have been too deterministic. This approach conflicts with general

learning. The application of statistical techniques can be used to move from a deterministic approach to a probabilistic one. Additionally, real-time autonomous action in complex and dynamic world environments is definitely going to be subject to a number of aspects of uncertainty. There is no doubt the soundly based and well developed statistical theory will be very useful in decision making and learning for autonomous mobile robots with these hypothesis assumptions.

While this author believes that there is tremendous potential for statistical assistance in this field, this point is not the focus of the research. Instead, the existence of specifically applicable statistical methodologies is assumed. This is all to say that wherever statistics would seem to be of use, we will simply acknowledge this fact and assume it is possible to create such a capability. Statistics will perform as a sort of black box for our purposes. In this way, we can focus on a more general overlying design view as opposed to a much more focused and specialized view. This will be much more advantageous to the accomplishing objectives we will set forward.

Chapter 4

Goals

The ultimate success in this area of research would be to create a robot which behaved just as the organic animals we co-exist with. This would be a machine that can cope with an unstructured environment through adaptation to that environment. The creatures of our world are able to do this. This is because they are equipped with all the necessary components to handle their environments. From that point, they perform learning on their own with no real guidance required. The creatures were made so that they could perform all the necessary learning on their own. This is the objective in the category of autonomous mobile robots. This is the ultimate goal. It is exactly what is of interest here. It seems that all the grand notions related to such advancement can soon be achieved. It appears that all the pieces are in place for such an accomplishment. But the advance has not yet been made. This research would like to provide a step towards this goal. This research has developed a general approach to the process of creating an autonomous real-time mobile robot.

4.1 Intelligent

AI is a very active and applicable field of Computer Science. Creating a sophisticated AI for any number of purposes is a very exciting and impressive undertaking. In several specific instances in the AI field this goal has already been achieved. For instance, chess AI is now equal in ability to that of a human player! An on-going point of interest in the AI field which has spilled over from the advances in robots is the objective of creating a sophisticated autonomous mobile robot. This is important in AI research because creating an advanced robot requires an extremely capable controlling mechanism which will oversee the robot. The mechanism that is required here is an advanced AI controller. As already mentioned, this objective is not yet complete. If it were, we would see the effects and ramifications which would ensue from such a completion. But we don't. Where are they? They are not yet here because the advance has not yet been made. An autonomous mobile robot AI controller is a major target of researcher's efforts in this area [12]. This is particularly what this research is interested in. However, making an amazing landmark step forward in AI is

not the goal here. Instead, we desire to build a framework into which the current state of AI research can be infused in order to achieve intelligent autonomous behavior that will exhibit advance decision making capabilities. This research aims to add to this tradition of AI research by working to develop new approaches for applying and incorporating advanced AI techniques in robots in order to allow the robots to perform sophisticated tasks.

This is the first goal of this research: This research will develop a framework in which to place existing advanced AI systems in order to allow the addition of intelligent decision making behaviors that are required for a system to accomplish a set of sophisticated goals.

4.2 Autonomous

Unfortunately, the majority of the successful ventures in AI involve relatively simple situations. These involve relatively uncomplicated and straightforward problem spaces. Further, these cases generally follow a well defined and known set of rules. But perhaps these situations may become very complex very quickly. Again, consider chess. This is not to say that these AI programs are unimpressive. Just that previously, the AI that has been developed for these cases is generally completely dependent upon the constancy of the problem space it is designed for. The AI is not made with the intention that it be able to handle drastic changes in its environment. Because of this, these particular AI attempts cannot maintain their functionality in a dynamic system. If they were to be able to cope with a changing environment, they would need to be autonomous [12]. This means that the AI would need to be a completely self-contained mechanism that was capable of making fundamental adjustments to itself in terms of its own behavior [12]. It would make these changes in response to changes in its environment in order to maintain its functionality [12]. There are many autonomous biological systems which can do just that. These systems are able to survive in places where the rules of the environment are subject to change because they can adjust to the changes. The robots that are of primary interest here will be those designed to operate in new and unknown environments. The rules in these environments will not necessarily be known in advance. The robots that encounter this type of environment will need to be autonomous in order to deal with every possibility. This is then a particularly necessary component for this research. Without this component, AI development is very difficult.

Adding this component will add further complexity. But perhaps this will actually make AI easier in the long run. Creating an autonomous agent will pass many of the problems of advancement and intelligence onto the robot instead of requiring the designers to deal with those issues. Time will tell.

This all related to the second goal of this research: This research will develop an approach to AI systems which will allow autonomous behavior so that even under changing and unstructured environments the system can continue to perform its function.

4.3 Real-Time

If the system was intended for use in a static environment it would not be important whether it was autonomous or not. However, the system will be intended for a dynamic real-time real-world environment similar to our own world. So the autonomous nature of the robot is a must. But there is another important point, likely the most important, that must be addressed for this very same reason. Because of this real-world environment the robot will operate in, it must be able to satisfactorily act in real-time in tune with its surroundings. The robot cannot ignore the events that are occurring around it because it has not yet finished processing the last event. It must not fail to respond appropriately to an event because the process of deciding how to respond takes too long. These things cannot happen or else the usefulness and well being of the robot will both be cast in doubt. Not only must the robot always superficially interact with the environment in real-time, it must also be able to adjust its behaviors in order to cope with changes in its environment in a way which does not hinder its real-time performance. The robot must be able to do its self adjusting in real-time since it must be able to actively interact with a real-time environment The primary reasons for this have already been stated. But, additionally it must not take so long to adjust its fundamental behavior that it is too late to be of any consequence. It cannot take so long to learn an important new fact that by the time the fact is incorporated into the knowledge base of the robot it is no longer of any use.

This leads to the third and final goal of this research: This research will produce an approach to AI systems that is capable of operating satisfactorily and adequately in a real-time environment.

4.4 Motivation

To be autonomous and efficiently accomplish goals, the robot will obviously need to be intelligent in its decision making. It must be able to learn and adapt through learning and accordingly adjusting its behavior in order to more efficiently achieve its goals. And it must be able to do all of this in real-time, or else it may not be practical in important situations. Some of the successes in AI have actually occurred in dynamic real-time environments. However, AI that does not presume a simplified world has not yet reached the level of sophistication that is intended for our system. There are currently no robots operating on Mars that are not dependent on the continual support of human instruction. If there were such a robot, it could act independently to pursue its objectives without the risk of human controller error, slow communication with the controller, or even loss of communication with the controller. All these risks can compromise the robot. This independent robot would eliminate the risk and expense inherent in the controller system. The robot would need to truly act in real-time or it could compromise itself by not responding to a danger in its environment in time. It would also need to be autonomous so that it is able to deal with the changing environments it would find itself a part of. And of course, it would need to behave intelligently. If an autonomous real-time AI was developed the way robots are used would be revolutionized!

Chapter 5

Methodology

5.1 Simulation

This research involves creating a model of the robot and a simulated environment to test the model in. The research will proceed through simulation. This seems particularly fitting given the disciplines involved. Therefore, the research will no longer be directly interested in the mechanical aspects of the robot, which were earlier given so much attention. While it was important to become aware of the capabilities of modern mechanics and sensors, this was only necessary to become familiar with the capabilities that can be expected from a mobile robot. This helps in considering what sort of designs can be developed and implemented as models. As the goals of this research have been stated, there is no immediate interest in engineering an actual robot at the time. If the approaches set forward here prove to be worthy of further study, the inclusion of real mechanical aspects of the robot would seem to be an immediate next step. But, this is meant to be an exploration into controlling an autonomous mobile robot. So, the focus will now revolve around that concern. Attention will be given to how this issue relates to the mechanical issues, but the specific mechanical issues will not be given the focus of attention. The robot, its capabilities, and its controlling mechanism are all to be simulated in code.

5.1.1 Negatives

As is to be expected, approaching the project through simulation does have some drawbacks. To begin with, it is a simulation. It is not the real thing. Producing a simulation does not do very much to further practical robotics in our world. And even more so, just because the program runs fine in the simulation, does not mean it will behave similarly if it is actually implemented. In a simulation the behavior of every object is fixed as the designer envisions [3]. It is built to behave in a certain fashion. Everything is thoughtfully planned by the same designer who built the robot [3]. This is definitely not the case in the real world, especially with the exploratory robot. The whole goal of this type of entity is to discover something new. Sensors simulated in code are

especially prone to errors of this nature [3]. It will be very difficult accurately reflect the true nature of the environment. Because of all these things, it is very difficult for a simulation to fail [3]. Also, because of the nature of a simulation, hardware defects that occur in the real world likely to remain unaccounted for.

5.1.2 Positives

Nevertheless, it seems that in a situation such as this paper, this approach will be a good choice. This paper is exploring a new approach to AI control of mobile robots. The object is to see if this approach has any merit. This requires a simple verification as to whether or not this methodology is at all worthwhile. The objective is to supply some sort of proof of concept. The most efficient way to reach this goal is through simulation. A simulation is altogether a much simpler task than a putting together an actual robot. The number of factors that must be considered is extremely reduced through this approach. Adding and adjusting additional features will be unproblematic through programming, while it would not be so in real life. Additionally, it will be only a matter of programming to create the desired environment, in which many interesting scenarios can be tested. In the real world, it can be time consuming to get statistically significant results in real robots. In this regard, simulation is much more efficient. It is also much cheaper to run a simulation. There are no hardware cost constraints. A simulation allows the AI component to be studied independently. This avoids many of the mechanical difficulties that arise which are unrelated to the AI component.

5.2 Distributed Intelligent Agents (DIA)

The underlying foundation for approaching the problem of developing an autonomous mobile robot is to use as much computing resources as possible while not complicating the system too much. We have a lot of capabilities with computing today. It seems silly to waste it. If it helps to use additional computing capabilities then it should be done. This pervasive fact of this approach can be summarized in the following way. This research suggests that to deal with all the issues involved in creating an intelligent robot, a good approach to beating the problem is to throw all the processing capability possible at it. This seems almost comical, but there is some seriousness associated with this statement. For instance, intelligence is a main goal. Why limit the robot to a single processor and limited memory to accomplish this task. We want an autonomous robot.

This will involve a tremendous amount of learning. Again, why limit the computation capabilities in this problem. We want something that is real-time. This is going to be hard to achieve if all of the other goals of the robot are to be met. In order to do this, we need to be able to perform all the necessary functions as fast as possible. Again, it seems silly to limit the robot given these constraints.

Creating an autonomous real-time robot that is sophisticated enough to operate in a dynamic real-world environment is a tremendous task. Man cannot produce a single processor powerful enough to effectively control a complicated autonomous system in a complex and dynamic real-time environment. So limiting the robot AI to such as design is a ridiculous idea. But notice that if the sub-functionalities of the system could each be independently operated in real time by a devoted agent, then all the sub-functionalities could operate as one in parallel in real-time. The real-time behavior of this system as a whole would depend on whether the sub-functionalities could be coordinated in real-time. A DIA entity is a task-oriented entity whose functionality is distributed among individual intelligent agents [10]. Each of these agents will have its own processing power to carry out their functional actions based on inputs, a knowledge-base/inference-engine, and general objectives. The system is divided up into components. Each of these can perform their tasks somewhat independently of the other parts of the system. They are given the power to do work on their own. The goals of this project will proceed under the frame work of a DIA since this appears to be a potential way in which real-time autonomous performance could be achieved.

Needless to say, developing an entire real-time autonomous DIA system is a tremendous task. This project does not presume that it alone can complete this entire undertaking. Instead, by setting some suggestions and guidelines, the research hopes to advance this approach to autonomous mobile robots. This will contribute to a crucial aspect of the DIA system by focusing on all the goals that have been set forward in order to bring the completion of an entire DIA system a closer reality. This research is an accomplishment on its own, but also a link in the chain of steps towards the completion of a DIA system. We will focus on the outline of the system as it relates the goals that have been aspired to.

5.3 Modularized Commands

It is now is a good time to cover an important approach this program utilizes related to the DIA structure. The robot will take a *high level* approach. This means that

the robot will not deal with the very low level details of its any of its actions, such as movement. All the commands the robot will deal with will be *high level modular commands*. They will be general commands not involving step by step execution instructions. Instead, they will be general commands, such as move forward. This is a general extension of the DIA architecture. By breaking apart sub-functionalities and distributing them to independent processing agents, the menial details of each agent can be hidden from the others. Instead, the agents only know the *high level* specifications of any other agent. This simplifies the development of individual agent units, because they can now interact with each other on a higher level.

The commands which are dealt with will be non-atomic commands that the mechanical parts carry without exposing all the involved complications that the actual execution entails. The commands the robot wishes to execute will be sent to independent agent units which execute the instructions self-sufficiently. These controllers manage the action completely independently. When a new command is given to these units, they handle all the required transition adjustments and produce the new action. If they are currently executing a command when a new command is issued, they transition to the new command on their own, without intervention and direction from the controller of the robot. As far as the robot is concerned, these actions are very simple, while in actual fact, they can be as complicated as desired. It's just that their operation is hidden in a different layer from the robot. Developing methods for which an agent can apply learning to its low level processes is not considered here. The actions are completely the responsibility of a separate processing unit. This allows the individual functionality of the independent agents to be completely self-contained.

Chapter 6

Guidelines

6.1 Approach

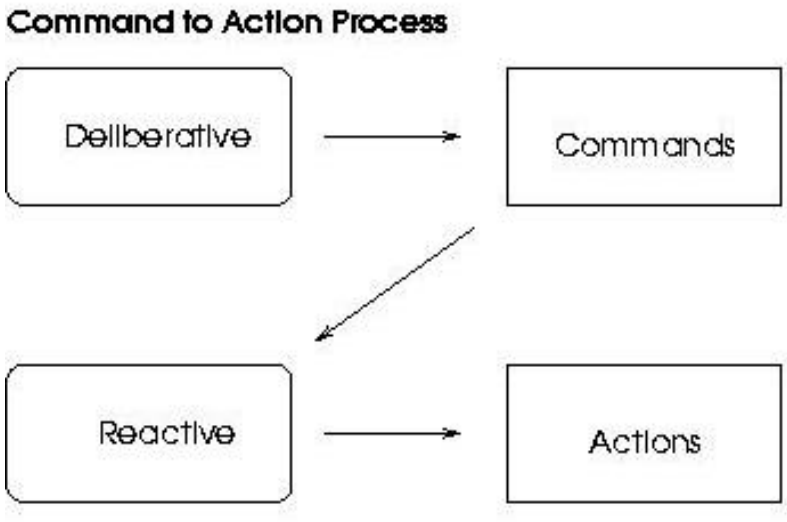
It is now appropriate to discuss the general approach suggested by this research. This section will begin the presentations of guidelines recommended in the creation of an autonomous mobile robot. The robot will operate in its environment under the influence of two distinct types of controllers. One will be the *behavioral* controller, and the other will be the *deliberative* controller. Each of these systems will embody an independent processing system. In this way, the *behavioral* approach can be followed while the *deliberative* approach can be used in the background. When the robot encounters a situation that it must immediately react to, it will. And when there is no such requirement, it can determine on its own what to do. So the commands to the robot are generated in a number of ways. Collisions of two possible choices in this regard are simply handled with priorities.

6.1.1 Reaction Side

The *behavioral* controller will be reaction based, and will provide necessary real-time interaction with the environment. This will be driven by input directly from the environment. The actions performed by this component will be behaviors that are triggered by environmental states. This type of input driven robot is common, due to Brooks' ideas [1] [3]. The advancement suggested is to allow the types of inputs which trigger responses be added, adjusted, and removed as the robot sees fit through learning. This will allow for an adaptable robot. The reactionary paradigm will be followed, with the added bonus that the ways in which the robot can react is adjustable. This approach is the main focus and will be discussed almost exclusively in the following pages.

6.1.2 Deliberation Side

When the robot is not under the pressure of immediate required response, the *deliberative* controller will step in. It will determine how the robot will proceed. It can generate commands on its own, and activate them. In this component, the notions of preference and judgment based on feeling can be made. This level embodies some sort of general will. Because of this, this system my override the reaction system in situations which it feels compelled to do so. This level will not be considered further in much detail. As far as this research is concerned, this portion of the robot control can be thought of as an aspect of the robot control which just does different things. Basically, the robot just loves to do things when not involved in the reactionary mode. It can go about behaving as it sees fit. It develops on suggestions for the robot's actions and performs them so long as they don't put the robot into extremely objectionable situations. The planning paradigm will be followed since the robot can think about what to do when under no pressure.



The deliberative unit gives suggested commands. The reactive unit Adjusts these commands as necessary. The result of this is the actual Action which is performed.

6.2 Assumptions

When the robot is first built, the designers are aware of its structure and nature. They know about its strong and weak points. They also know what the robot will be intended to do. It is apparent that for the robot to be autonomous it will need to be suited

and designed for tasks it is to undertake. In the design phase of the robot's development, the general plan for how to equip the robot will be made. The robot will need to be built for the tasks it is meant to perform. In order for it to behave effectively and intelligently, it will need to possess the correct set of base abilities. In the same way, the robot will need to be equipped to understand its environment in order to accomplish its objectives. It will need to be suited with the correct sensor capabilities to perform its objectives.

Once all of this design has been adequately completed, the task then is to allow the robot to use all of its functionality in an intelligent manner on its own. The robot is given all the tools it needs for its tasks, and so now it must manage all of these tools in order to do its task. But further, it will need to be able to detect occurrences that are related to it fulfilling its tasks. It must be aware of its weak points in order to maintain its safety. It has to know when it comes into contact with a hazard. The designers must incorporate the detection of events that are important to the robot's goals and well being. With this done, the robot is equipped to handle these situations. If it is not, then it is of no use for its intention.

6.3 Sensors and State

As the above suggests, the robot is bestowed with the sensors it requires for its objective. Each of these sensors monitors a specific aspect of the environment that is relevant to the robot. As we saw, there may be a variety of sensors. Each sensor reads the environment in the cycle it is designed to follow. The sensors read at a very rapid rate. We can assume that the rate is relevant and useful in real-time, or else the sensor would not be utilized. Some sensors will give immediate feedback, while others will need an extended period of sensor processing in order to be beneficial. At anytime, a sensor retains its current reading. The sensors cycles may, or may not, coincide exactly. But at anytime there is still a measurement available in every sensor. At any given time, the current status of all the sensors indicates the *state* that the robot is currently in.

At this point, a nice and useful classification of sensors can be made. This is a distinction which can accurately be based upon the measurement of the sensor as either input or output. The input sensors measure things that are attributes of the robot's environment. These are pieces of knowledge that can be used to help the robot operate better in the environment. The other category is the output sensors. These measure the actions that a robot does. The robot can then be made aware of what its actions do and

can use this information to better act in the future. The key with this category is that these sensor readings are generated directly from the robot itself. Thus, they are output.

In the first group of input sensors a slight sub-classification can be made. This involves measurements directly from the environment, and those that are actually taken from the robot. Measurements from the robot are from sensors that indicate the status of a certain piece of the robot. But these are not actions taken by the robot, and so are still categorized as input. These distinctions will become useful further into the paper.

The notion of a sensor reading brings several issues into play. These initially appear to be new issues which must be individually studied. These are notions such as velocity and acceleration of a measurement. On closer inspection however, these need not be a special case. If needed, one sensor can provide its measurement, as well as its velocity and acceleration measurements. These can be simply computed from the original sensor measurement. They can then be treated just as any other sensor on the system. They can be used in just the same way as the other sensors and need not be considered as a special case.

6.4 Laws

Just as the robot has been equipped with all the components necessary for its tasks, it is equipped with intrinsic instincts related to its purpose. The robot's creators instill the required intrinsic characteristics, and the senses to be aware of them. They cannot be learned or adjusted. They are the nature of the robot. These are the behaviors that the robot automatically submits to. They are the reaction behaviors. The robot will learn how to automatically react to these occurrences.

There is a huge static table of all the laws. It will be partially distributed to parts of the robot which require parts of it, as will be demonstrated later. The distribution schema can be used to develop redundancy in the system, which will help safeguard loss of necessary data.

Template Law Structure

Sensor ID	Law ID	Reading	Less/Greater	Meaning

Sensor ID is a number assigned to a sensor which uniquely identifies the sensor involved in the law.

Law ID is an identifyer of the specific law in question.

Reading is the measurement the the sensor which is the critical threshold point related to the law.

Less/Greater is an indicator bit which identifies whether the sensor values which are above or below the threshold point are the critical values making of the critical region.

Meaning is a score assigned to the law which indicates the measure of like or dislike the law has with the robot.

Just as humans do not learn what the sensations of tickling or pain feels like, these laws are the built in distinctions of the robot. These intrinsic traits are called *laws*. These are certain *states* of the environment that have particular implications for the robot. To the robot, they are simply a measurement range from a specific sensor, or combination of measurement ranges from several specific sensors, which carry with them a preference of the robot. Some sensors will be more active than others in the *laws.* This preference is a score of measure of like or dislike. It denotes whether the robot associates pain or aversion to a specific *law,* or whether it associates satisfaction or pleasure with that *law*.

Combining Laws

Laws may appear a single template:

Sensor ID	Law ID	Reading	Less/Greater	Meaning

Or, they may involve a series of law Templates:

Sensor ID	Law ID	Reading	Less/Greater	Meaning

and/or

Sensor ID	Law ID	Reading	Less/Greater	Meaning

and/or

○
○
○

and/or

Sensor ID	Law ID	Reading	Less/Greater	Meaning

6.5 Events and Critical Events

If the robot's current *state* indicates that a *law* has been reached, we are in a situation which will be defined as a *critical event.* Because the robot knows the nature of the *law*, it will know whether or not it is attracted to the current state of the sensors. The robot will then need to immediately react accordingly. How the robot will act accordingly will be discussed further, later. But for now this can be understood as a reaction that is automatically generated by the robot given its state. It is environmental input driven.

There is another sort of situation which is related to the *critical event*. This situation will be called simply an *event.* This is a *state* of sensor readings which in some way connected to the *critical event.* They are part of a pattern we recognize as leading up to a *critical event*. If the *critical event* is negative, then the *event* gives information concerning how to avoid it. Or if the *critical event* is positive, it will give information concerning how to achieve it. There is a way to behave in an *event* state which will help meet the goals in the *critical event state*. The desired action of the *critical event state* honored through actions in the *event* which honor the meaning of the *critical event*. An *event* is treated as a reaction state. Given that an *event* happened, the robot immediately responds to that event. The robot is driven by the event.

A good visualization of what this looks like is a web or tree of *events* and *critical events.* Next, the creation of such an object will be discussed. The table of possible important states is a list similar to the *laws* table. But this table is a dynamic list. It is very subject change and growth. *Events* are learned in order to help the robot behave better in given situations. This is a big table which is also partially distributed to add redundancy.

Template Event Structure

The Event Structure is the same as the Law Structure

Sensor ID	Event ID	Reading	Less/Greater	Meaning

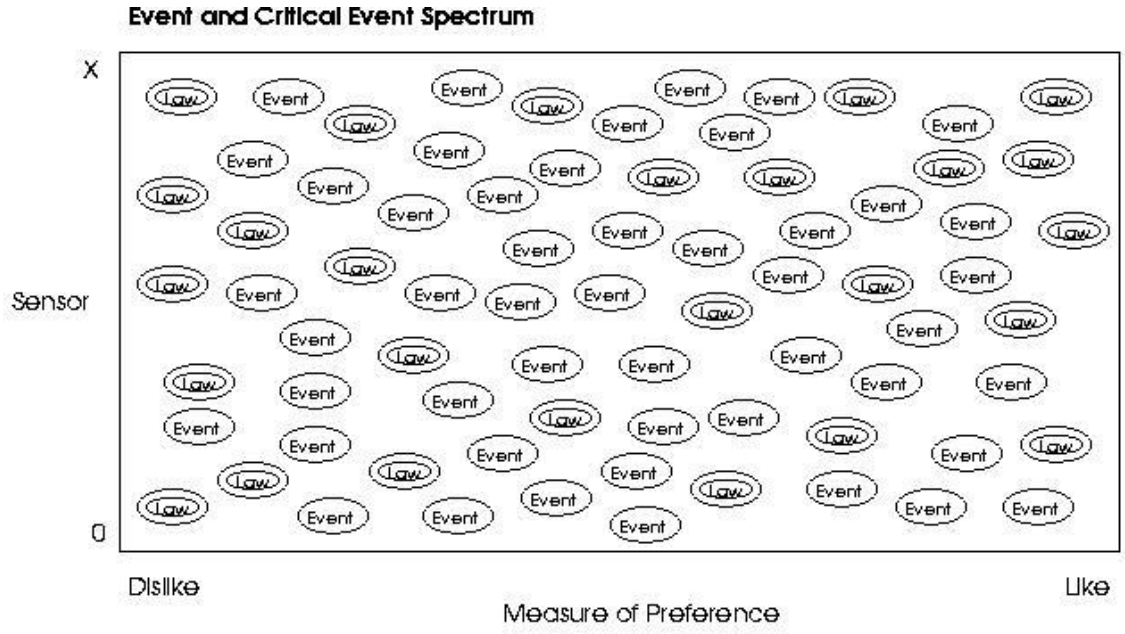
vs.

Sensor ID	Law ID	Reading	Less/Greater	Meaning

The difference is that laws are static and fixed beforehand. While Events are dynammic. They can be added, adjusted, and removed. Therefore the identification used in laws is static, whereas that used in Events follows a dynamic system.

The rules will be combined in the same way as the laws were in the previous figure.

From now on, *critical events and events* will be referred to simply as *events.* An *event* can be classified into three categories. There are *environmental events*. These involve *states* related to the environment. Examples of these are water present, gravel type, and slope degree. There are *entity events* which are *states* involving other agents in the environment that are dynamic to the robot in that it can interact with them. And there are *preservation events*. These are *state* condition necessary for the robot's operation such as heat, radiation, and pressure. Naturally, some sensors will be more active than others in the *events.*



Of all the states that the robot can be in, there are some which have been designated as laws that the robot must adhere to. There are other states which are relevent for predicting these Laws. These are Events. The robot generates these in order to perform better. They are based on certain sensors and have attatched to them a measure of preference. The Events are dynamic. They may change in this figure with time. The Laws on the other are fixed and so will not change in this picture.

6.6 Learning Events

The sensor readings are made available to be read, but they are also forwarded to a chronicling device which will keep records of everything which happened to the robot. This is a sort of history that is used to learn which *states* are actually *events.* Just as the robot is given the correct mechanical modes for interaction with its environment and is given the correct sensor array to allow it accomplish its task, the robot is pre-equipped with all the necessary algorithms to produce these advances. These

algorithms are another set of intrinsic endowments that the robot is based on. The robot is not designed to learn how to learn. It instead already knows how to learn.

The primary vehicle for this learning will be pattern recognition based on the sensory inputs. But, this learning is done before any real consequences can be met. The things which need to be known are inferred whenever possible. The robot cannot learn from being destroyed. Initially, the robot is aware only of its laws. Then learning and inference algorithms begin to learn new *states* which are *events*. These are additional *states* which are related to the *laws* as we said. At first, the good and bad *laws* make up the whole spectrum of *states*. Then, in between the *laws*, *events* begin to appear. The robot starts with only its *laws*, and it advances its understanding to effectively meet the *laws*.

In considering learning in the robot it is important to now note the nature of different kinds of learning which may occur. There are cases in which we need to learn in spite of sensor inaccuracy. There are cases where the sensors can become predictors and an *event* can be learned. But then, there will be times when the sensors do not predict anything, but a change has occurred and must be learned. In these cases a second *event* may replace a first event. When the behavior seems to switch back and forth between these two *events* the event in use must be switched back and forth. All of these must be handled by the inference algorithms. All of these learning possibilities can be broken down into two classes. The first class is the events which are not under the control of the robot. These are acts of nature which must be understood. The second class is the actions the robot can control.

6.6.1 Environment Learning

For learning about things which are not directly caused by the robot, there will be dedicated learning algorithms. They will learn from the sensor information if any states require special consideration. This will proceed through statistical modeling. This will be particularly useful because of noise with the sensors. Statistics will provide an established way to deal with the variation. Samples concerning a specific occurrence can be generated from the sensor readings and in order to determine if there are predictors of that occurrence. If there is a predictor, this predictor will be used to provide the robot with additional information concerning the *state*. This is now an *event,* and we can use the information we learned from it to act better in that *state*.

6.6.2 Self Learning

The robot needs to learn what its actions do. In this way, it can it can perform the action which is best given the current *state* the robot is in. Also, it can detect *events* based upon the action. By knowing what an action does, the action can be related to all the *laws.* Because the result of the action is known, the result of the action in regard to the *laws* can also be known. If an action given the current *state* will result in a negative *law* being reached, that action should not be performed. The current *state* the sensors are in alerts the robot that it is in an *event.* The robot knows some information about how to act given the state which adheres to its goals. In this case, it knows it should not perform the action because it will result in a negative *law* breach. By knowing what an action does, an action which leads to a *critical event* can be determined and an *event* can be made out of that. In these *events* the robot can act in accordance with the related *laws.* The value of the action determines the *state* of the *event.*

When an action is performed by the robot, the result can be seen in the sensor readings. Specific algorithms will learn what the actions of the robot do. Once the result of actions becomes known, the correct action can be paired with specific states. When the robot is in a *critical event,* the move to do can be known since the actions results are known. All of this must be done on the fly and be robust to all the complications which may occur.

By having the robot constantly know how its actions effect the environment, it need not worry about changes in the ways its actions perform, because these will become clear. Further, no special actions must be taken in the case of a change, because by knowing the actions results the best course of action will always present itself. The complication is that each time an action begins behaving differently the *events* related to that action are no longer applicable, and a grand overhaul of all the related *events* is required.

6.7 Rules

Rules are the way the robot should act in a given *state.* They are the reaction behavior that is immediately activated by the correct environment conditions. An *event's* knowledge of what actions do will allow certain actions to be preferred or rejected. This is how the special states that we talked about allow us to perform better actions as hinted at earlier. If we reach an *event state,* we know how to best respond in that state because

of what the responses will mean as far as the *laws* go. The *rule*, or behavior, is fired.

The *rules* force behaviors that satisfy the objectives of the *laws.* Each special state may have several *rules* attached to it. These specify the actions that are or are not desirable in the given state. The *rules* correspond to the special *event* and *critical event states.* The rules will be given a level of importance based upon the *critical event* they are related to. *Rules* may also appear as a parameterized unit based upon other environmental inputs.

It is clear that this will produce a large table of rules. These will correspond to the table involving the *events and critical events.* There will be a correspondence between the related pieces. As these tables correspond in a one to one manner, the *rules* will be a very dynamic table. It will be very dependent on whether the actions of the robot continue to behave in the same manner. If an action changes, the rules will all be subject to the change.

Template Rule Structure

Sensor ID	ID	Action Command Implication

A rule is related to an event. To find an Event, all that is required is the sensor related to the Event, and the specific Event on that sensor. This is whether it is a Law or an Event

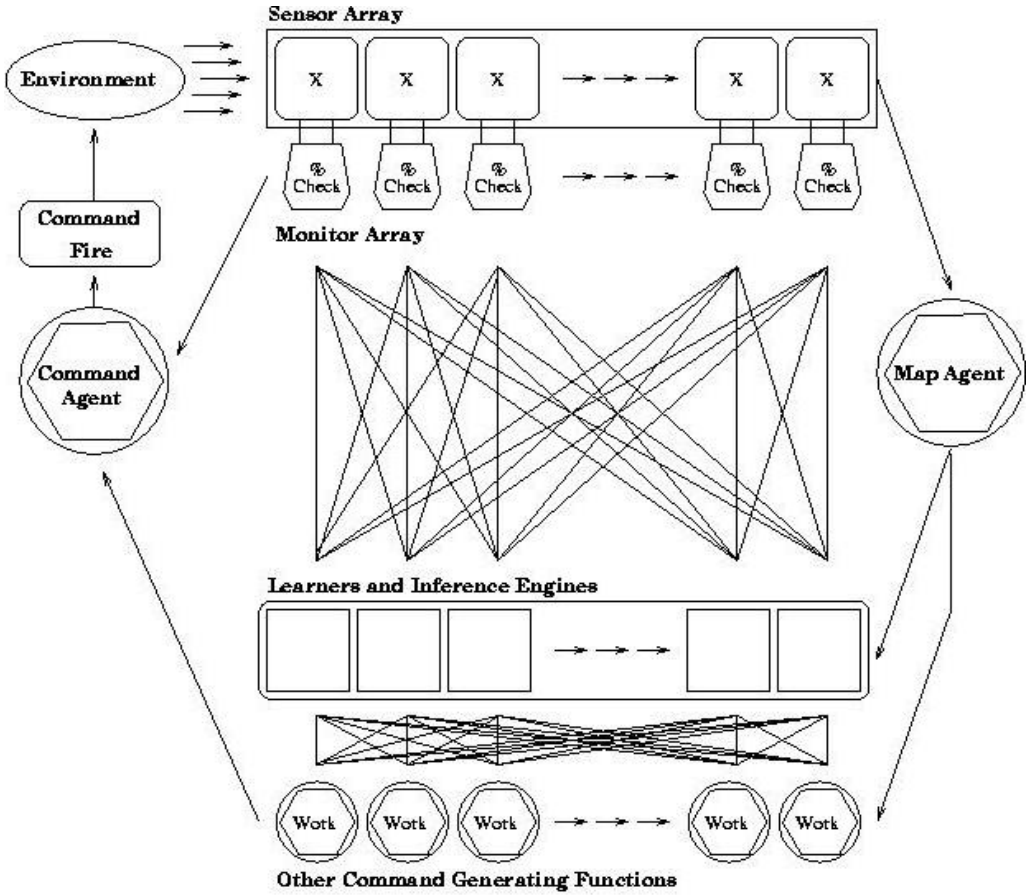
The resulting effect is given the the action Implication. This Instructs how to proceed given the State.

6.8 Monitors

So far, there are different *events* which the robot is aware of, and attached to these events are the *rules* of that *state.* The sensors only read the information from the environment; they do not have any specific capability to detect whether or not they have entered and important *state* which is relevant to an *event*. There is a separate unit which will take care of this by watching the sensors to see if they are in an important *state.* These are the monitors. All the *events* have sensor ranges which indicate the *event*. The monitor is aware of all of these ranges and detects them. Some *events* require separate sensors, though. So the monitor is simply detecting important *states*. The robot doesn't want to be interested in unimportant background noise that comes in that the sensors have picked up. It just wants the important information. When a relevant *state* is detected, the system will be notified. It will be determine whether this along with

all the monitor input constitutes an *event*, and if so the appropriate rules applying to the *event* will be used in order to produce an acceptable action. This will be discussed soon. It is important to stress that this is how immediate reaction is implemented. As soon as an important *state* is encountered which might involve an immediate behavior reaction, the system is thrown into gear to respond accordingly.

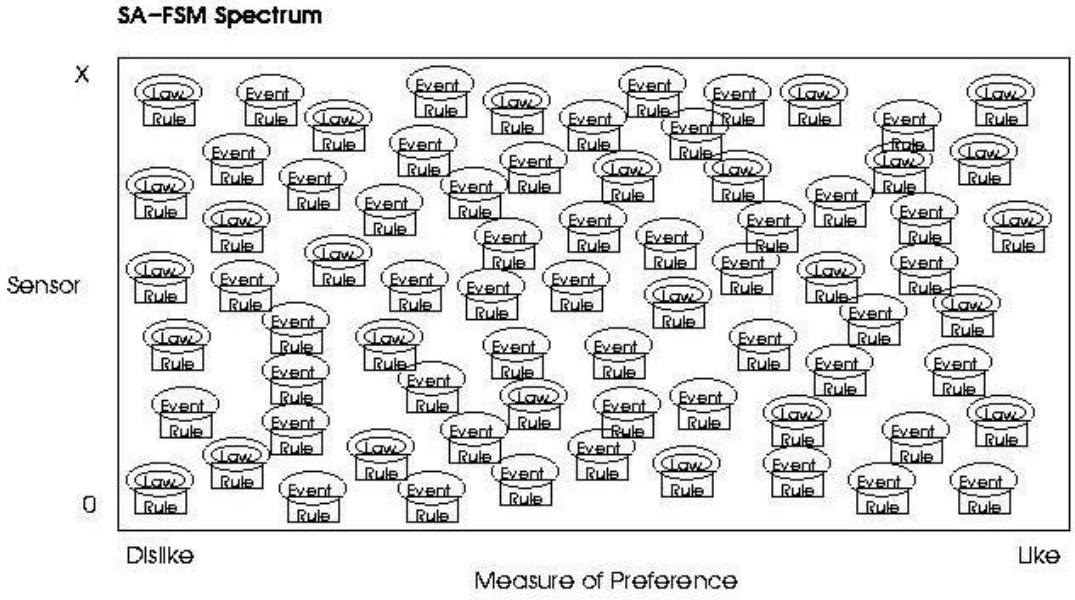
As the *events* are developed, the *events* a monitor checks for are changed and adjusted. These changes come from a number of algorithms which are learning the different *events.* Each sensor is being checked for predictive power in certain situations. Different types of algorithms are doing these checks. It's not just one algorithm for one sensor. This is because the sensors can be important individually, or they can be important as a group, as far as *events* are concerned. The monitors need to always put a priority on checking the sensor. But, they need to be updating there list of what to check too. This is where separate processing capabilities can come in very handy. Both of these tasks can be done at the same time. The trick will simply be to coordinate everything which will soon be discussed.



6.9 Self-Adjusting Finite State Machine (SA-FSM)

The place where the monitors pass on the interesting *states* they come across is the SA-FSM. This is the compilation of all the *events* and corresponding *rules.* The base construction involved in this unit is a *finite state machine*. This type of a machine encompasses all forms of computation, so it is a good unit to start with. The addition here is that this machine may be adjusted. An SA-FSM has an addition operation which is a module that adds, removes, and adjusts the states of the FSM.

The FSM is all the *event states* and the responses to those *states*. Each entry in the SA-FSM is the *events* that were discussed earlier. As stated, these *events* carry with them sets of corresponding *rules.* These make of the *states* and resulting *rules* which comprise the SA-FSM. The picture of the FSM is identical to that of the *event states* the robot has learned.



The SA–FSM has all the reactions for important states of the robot. The Event type of reaction may be added, adjusted, and removed in this picture. Given a State, the Rules that are to be applied to the state are.

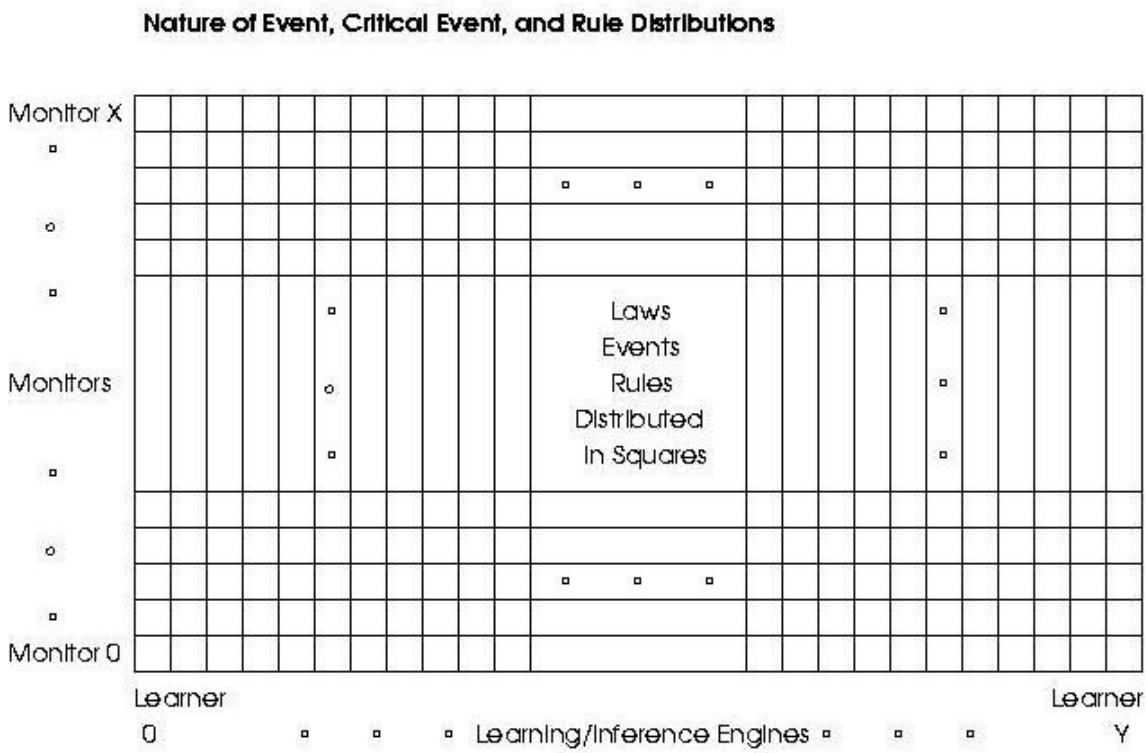
The SA-FSM receives the states from the monitor, and determines if an *event* has been reached. If so, it will know the *rules* that correspond to the *event* and will quickly apply them to the action to be taken. As the all the monitors send along *state* warnings, the SA-FSM checks to see if an *event state* has been reached. If so, all the

rules applying to that state are brought into effect. When an *event* is encountered the FSA executes the behavioral action associated with that event. Given the *state event,* there are certain actions which are preferred. This is the fast *behavioral* portion of the project. This does not account for all the types of actions the robot will want to perform, just those that are important in real-time operation. This cycle involves receiving input, detecting the state, finding the procedure given the state, and acting to adjust the state.

6.10 Adjusting

While the robot is in operation it is constantly learning. It is learning about the environment and the things in it. It is developing new states which it knows an advantage in regard to its actions. If the results of some of its actions change, the results that it knows about in other *event states* need to be adjusted. If an action no longer performs in the same way for whatever reason, the robot needs to adjust its knowledge based on this. This may be predictable and so an *event can be made which* signifies an occurrence of a change. But, perhaps it cannot be predicted. It just seems to happen. Then all of the old information regarding the previous behavior is no longer applicable. It may need to be saved off and things switched to reflect the new behavior. Then, it may change back without an indication of why. The switch must be made back again. This is a sort of context switch that may occur. The point is that there will be many different forms of learning of *event states* going on. This will result in a lot of necessary adjustment to the *events.*

The adjustments need to be done quickly and quietly, behind the scene. *Events* based upon actions change if the actions change. Many components in this system use this *event* and action specification. When it changes it needs be changed in all the places it existed. It needs to be changed in the learning algorithms that used it. It needs to be changed in the monitors that look for it. It needs to be changed in the SA-FSA that responds to it. Constantly updating everything in a coordinated synchronous manner is imperative. It adjustment must happen in unison in parallel. This is because one part of the system cannot be changed while the other is unaware of the change. If this were to happen, the two components would be unable to communicate to one another. Learning and inferring these changes needs to happen quickly, and must be applied just as quickly. It will be a very dynamic system, whose order will relate the number of sensors.



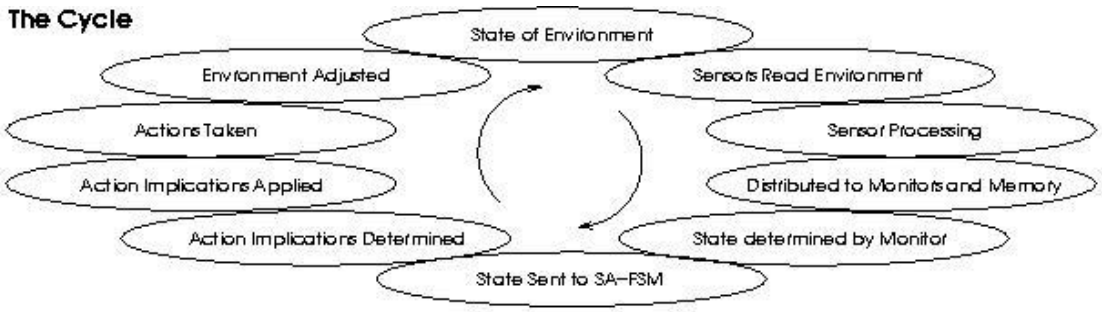
Laws, Rules, and Events are distributed out among these two groups of agents in the system. The distribution is not equal and uniform as this image may suggets, but rather the image is meant to show that the distributions of the information is discrete and somewhat disjoint. Each individual monitorand learning algorithm has the information which is immediately relevent in its task.

Each box represents a part of the information that the given unit has access to. Some information is overlapped in the monitors and the learners. There may also be information reapeted in each of the learners.

6.11 Actions

It is now appropriate to go into more detail concerning how actions are created and executed. At any given time, the robot will have received a suggested course of action from the will level of the robot. At the given time, the exact *state* becomes known. All the *rules* which are applicable to the state are applied to a generic action that is to be performed. The action desired by the will is then imposed upon the current state of the action. If it is feasible to perform the desired preferred action it is performed. If it is not, all the *rules* that are now applied to the condition of the robot's action are crunched together to create the possible actions. The *rules* which dominate this compilation of actions are those that are fired. The current *state* of the machine holds rules about important and relevant actions. These are used if they exist. Otherwise, we have a

general template for what to do developed on the will level of the robot. It is fine to run this command because there are no indications otherwise. The general will gives commands. The SA-FSM does what it can to follow these while following any reaction *states* it comes across.



6.12 Interrupts

To clarify the process, there is another concept which can be discussed. This is the concept of interrupts. Through the robot, there are many cycles occurring, and they may all be different. The different sensors read and pass along their input in different ways. The learning and inference algorithms proceed in their tasks in different manners. The SA-FSM receives a *state* and acts on that state. Updates are made to various components from learned information. We cannot force any of this to follow some pre-defined cycle. Because of the possible difference, the robot performs in an asynchronous fashion. We want the robot to appear as a synchronous unit though. All of these differences need to be handled beneath the surface. This can be done using a connected system which runs on interrupts. All of these pieces can function as their own processing unit. But information for how they should proceed can be passed around like messages.

Chapter 7

Experimentation

In order to conceptualize and experiment with the proposed ideas, a tangible testing ground has been developed and created. This project did not have at its disposal the necessary hardware components which allow a robot to interact with its environment. Because of this, an actual robot could not be constructed to implement the given approaches. However, this project did have at its disposal the necessary software creation capabilities. The software components are what run the robot, so the approaches were tested through their creation in software. Therefore, this research was able to develop a programmed system through which the ideas and concepts of this research could be considered. This allowed for testing and verification of the approaches that are under consideration. This work is described below.

To solidify the concept approaches, and to demonstrate the various aspects of these approaches, a system model has been constructed. This model emphasizes the key components of a SA-FSM robotics control system as it would function in action. The model was developed in order to develop, consider, and conceptualize the possible interaction paradigms for the components of the system. Additionally, this was necessary as a first step to developing a software environment in which to test the ideas. The specifications are given graphically in Appendix B.

The model led to the development of a software environment to begin implementing the ideas of this research. Upon completion of the abstract model, a simplified version based off of the model was implemented in java. The implementation reflects the overall architecture presented by the initial model. This implementation was used to experiment with the ideas of the research. The java source code is available in appendix C.

7.1 Model

This section describes the model used for the development of a SA-FSM robot. This includes the interaction between a robot of this type with itself and with its environment. Appendix B contains four figures that are a graphical representation of the model introduced in this section. These figures will be a very useful reference for the

preceding discussion.

7.1.1 Environment-Robot Relation

Please refer to Figure 1 of Appendix B for a graphical representation of the following discussion. The basis of the model developed for this research is a model environment. This is a necessary component because everything happens as part of the environment. The environment is built as desired in order to consider certain types of environments which we may wish to test the robot in.

There are three distinct types of components which comprise the environment. The first of these types is the state related aspects of the environment. These are the certain environmentally-related components of the environment itself. There may be any given number of these. They are the "nature" of the environment. At any given time the state of any given element of the environment is set at a given point. These settings may change over time due to processes acting on them and adjusting their levels, as will be discussed next.

The second type of entity which inhabits the environment may adjust the states of environmental components. This type involves agents. These may simply be processes of nature which act on the states of the environment described above. Or, they may be actual entities which peruse their own set of objectives and goals in the environment. There may be any given number of these. The states of the environment discussed above are adjusted by these agents performing their tasks in the environment. These agents themselves have a physical state in the environment at any given time. They adjust the environment which includes parts of the environment which do not belong to them, as well as the part of the environment that is themselves.

The final component of the environment is a single entity, the robot. Because this research is interested in testing an approach to robot control, this is the focal point of the model. The environment is built and designed based upon what is desired to test for the robot. The robot acts as a part of the environment. Its actions take place as an interaction with the environment. The robot adjusts its environment through its actions, as well as interacts with the other agents which inhabit the environment. Because the robot itself is a part of the environment, its physical state in the environment is also a part of the environment.

The robot itself is modeled with a series of components. The first of these components relates to the way in which the robot interacts with the environment. The

robot operates in the environment. The robot may operate in the environment in a number of ways. It does so with its actuators. These mechanically based components of the robots are the means through which the robot exists in the environment. The robots actions produce effects in the environmental states as it desires. The robot directs its actions in the environment by directing its actions which effect the environment. The actions also change the robots state in the environment. The robot becomes aware of its states and the states of its surrounding environment through its sensors. There may be any given number of sensors. These are the mechanical means through which the robot can interpret its world. When the sensors read the environment, they make the information they learned available to the robot for use in deciding its actions.

7.1.2 Robot Responses

Please refer to Figure 2 of Appendix B for a graphical representation of the following discussion. In the robot model, the robots interface to the environment was its mechanical components, as we saw. These are connected to the environment and the robot, and are the conduit for interaction between the two model components. The other model components of the robot do not directly interact with the environment. They are contained within the robot. These are the inner workings of the robot. The rest of the model is concerned with the interaction of these components. These parts of the robot are the processing components which determine the way the robot will interact with the environment. This happens separately from the environment. It occurs beneath the surface of the robot. It is hidden from the environment.

The first layer of these inner components is the robots environment interaction control mechanism. This has two parts. The first section is called a monitor. This is because this part monitors the sensors for important environmental states. There may be any given number of these. A mechanical sensor relays its reading to its monitor. The monitor has a list of important states which it watches for. In this way, unimportant states are simply ignored by the monitor, and so ignored by the robot. When an important state is reached, the second part of this first layer comes into play. This is the SA-FSM component. This component determines how to react to the given state that the monitor saw. This is expected to be done very quickly. There is no processing involved in this determination. The action is simply reference by the important state. This is a complex issue in and of itself. This component of the model is concerned with this issue.

This setup allows unimportant states to be ignored, while allowing for very fast reaction to important states. By having many monitors concurrently checking for important states, many states can be considered. If the SA-FSM component had to check for every important state, it would never react, it would have too many things to check. Many monitors allow everything to be checked. Then, at any given time the SA-FSM can determine what to do from a much smaller set of inputs. These being the important states the monitors came across.

7.1.3 Robot Knowledge

Please refer to Figure 3 of Appendix B for a graphical representation of the following discussion. The underlying foundation for the reaction portion of the model just discussed is the robots laws and rules. Each of these relate to an important state of the environment which the sensors perceive. For each of these important states, the robot learns the best way to interact and respond to the environment.

In the model, the laws are environment occurrences which are of direct importance to the robot. Rules on the other had are environment occurrences which are indirectly important to the robot. These are portioned out to the monitors to watch for. The monitors are interested only in detecting such a state. The SA-FSM on the other hand is aware of all the states, as well as the best responses to the states. The robot has special data structures which manage this information. This portion of the model is dedicated to the management and distribution of this knowledge. This portion of the model is based upon a distribution of these laws and rules. Additionally, the structure which holds them must be adjustable as the rules and laws may change with time. As they change, all the components which contain them must be synchronously updated with the changes. The model represents this as a single entity of information. Certain components have access to certain parts of this data. As the data changes or new data is added, the access is simply adjusted to reflect the changes.

7.1.4 Robot Learning

Please refer to Figure 4 of Appendix B for a graphical representation of the following discussion. All of the above discussion left out the fact that the underlying foundation is adjustable. The components which adjust the knowledge are preset learning mechanisms. They have their intrinsic algorithms for learning, as well as the

knowledge of past events the robot was involved in.

While the robot is running, the sensors relay all of their input to a memory location as well as to the monitors. The history of the robot is therefore available for consideration. It can be used as necessary. Memory management is the issue in this component. There is so much data and so many other components which will likely be interested using it. The distribution of this information is important.

The components which need to use this information are the learning components of the robot. There may be any given number of these. All of the algorithms which learn the reactions to states, as well as learn new important rule states are aware of certain laws and rules which pertain to them. They use these in the background to help learn. The laws are static, but the rules are adjusted by the learners. The laws are used, along with the memory, to determine new rules. Additionally, the memory is used by the algorithms to determine which actions are best in a given state. As described above, the changes to the knowledge base result in a new set of information for the components which use it.

7.2 Implementation

A simplified implementation of the above described model was developed in java. This allowed for the initial construction of a controller system for initial testing in this research. The file structure of the implementation is based on the model described above. The directory structuring which was used for the programmed code is a multilevel tree structure which houses various components of the implementation at each level. This structure was developed in order to facilitate the development and expansion of the system described by the model. A discussion of the nature of the structure as well as the implemented files which reside inside will now be undertaken. Appendix C which contains the code described below may be referenced during the discussion of the implementation in order to help follow the discussion. The following table shows the file structure layout.

Directory	Subdirectories	Files
theSIS	AGENTS, robot	ENVIRONMENT
agents		
robot	ai, fsm, mechanical	ROBOT
ai	LEARNERS	MEMORY
LEARNERS		POWER_EVENTS
fsm	MONITORS	FSM
MONITORS		POWER_SENSOR_MONITOR
mechanical	ACTIONS, SENSORS	
ACTIONS		POWER_ACTION
SENSORS		POWER_SENSOR

The base directory is named **theSIS**. It contains two subdirectories and the main program. The subdirectories are **agents** and **robot**, and the program is **ENVIRONMENT**. **ENVIRONMENT** is a class which contains all the state information in the world. It contains the code is a program which the cycle of the entire system runs off of. Everything which affects the environment does so in terms of its specific cycles. The effectors of the environment are found in the **agents** and **robot** directories.

In the **robot** directory we house the robots structure. This is found in the **ai, fsm,** and **mechanical** sub-directories. The class which encapsulates and uses this structure is found in the **ROBOT** program. This oversees and coordinates the overall interaction of the components of the robot, as well as their interaction with the environment.

The directory **ai** houses the deliberative components of the robot. These are the programs which learn from the environment and adjust the overall functional behavior of the robot. It has a subdirectory called **LEARNER** which house a learning mechanism related to the power functionality implemented in the model. This inference component is labeled **POWER_EVENTS.** The current implementation also keeps a database structure called **MEMORY** in the **ai** directory which houses the history of the robot.

The **fsm** directory houses the actual SA-FSM components of the implementation. These are held in the **FSM** program. Related to this functionality is the **POWER_SENSOR_MONITOR** held in the **MONITORS** directory. This is a component of the robot as a whole which is directly related to the SA-FSM functionalities as described above. This code monitors the input from the power sensor for important input states.

The **mechanical** directory houses the pieces of the robot which directly relate to the environment. These components are found in the **ACTIONS** and **SENSORS** sub-directories. These directories contain **POWER_ACTION** and **POWER_SENSOR,** respectively. Their functionality is very straight forward. The action does something with respect to the power of the robot, while the sensor senses the power levels of the robot.

This implementation is based upon this power functionality of the robot, and so the components of the code have been developed for this aspect of the robot. Adding additional features to the robot simulation will involve scaling the current implementation with the additional required components for the new ability. This is not particularly difficult as the code is designed with a very modularized paradigm. This simply means that new components are just dropped in as they are developed. The one piece of code which is not yet completely modularized is the SA-FSM. Additional work needs to be done on this component in order to allow a seamless integration of additional robot features.

Outlook

This appears to be a very promising approach to intelligent autonomous real-time behavior in artificial creatures. But is this approach really possible? Through the above model, it has been demonstrated that on a smaller scale it is possible to implement such an approach. But is it feasible to extend this into more complicated realms? I believe that the answer is yes. Of course, the effort that will be required for such undertakings will increasingly grow as the size of the project grows, but there are certainly sufficient computing resources available for this methodology. The point which may cause this approach to become infeasible would be if there is more required communication among the components of the system than can be handled in real-time. Special care must be put into insuring that this problem does not become a reality. I do believe that this would be possible to accomplish.

The reason such an approach has not yet been fully implemented is due to the sheer complexity involved in the creation of such a sophisticated system. Indeed, there is not yet any system, regardless of methodology, which can accomplish the goals set forward in this work. Fortunately, it is almost impossible to argue that approaching the problem in such a distributed way as proposed here won't reduce the complexity of developing such a system. The primary drawback of this approach is that it is based upon the characteristics of a given robot. Many of the concepts put forward by this idea are applicable to any given robot, but a specific instantiation of this methodology will not be immediately reproducible on a different type of machine.

However, as the robot industry becomes more and more standardized, perhaps developing individual robot components based upon the approach suggested here would produce a sort of plug-and-play structure in which a standard set of robot components could be operationally combined very simply in many ways. With such a system in place, the inclusion of a new component type would appear to be fairly straightforward. In essence, this approach would develop into *the standard* for robot components and their assembly. In the end, this would produce a means for the mass production of robots.

References

[1] Joseph L. Jones, Bruce A. Seiger, Anita M. Flynn. Mobile robots: inspiration to implementation. 2nd ed. Natick, Mass. : A.K. Peters, 1999.
[2] Menzel, Peter, Faith D'Alusio. Robo sapiens: evolution of a new species. Cambridge Mass.: MIT Press 2000.
[3] Nehmzow, Ulrich. Mobile Robotics: a practical introduction. 2nd ed. New York: Springer, 2003.
[4] http://www.unece.org/press/pr2004/04robots_index.htm
[5] http://www.ifr.org/
[6] http://www.darpa.mil/
[7] http://www.robocup.ort/
[8] http://www.trincoll.edu/events/robot/
[9] http://marsrovers.jpl.nasa.gov/home/index.html/
[10] Maurice Eggen and Gerald Pitts, *Distributed Intelligent Agents: A New Approach to Distributed Artificial Intelligence in Robotic Systems.* (PDPTA: 2003), 589-595.
[11] Rodney A. Brooks, *Intelligence without representation*.* (Artificial Intelligence Volume 47, Issue 1-3: 1991), 139-159.
[12] Hani Hagras and Tarek M. Sobh, *Intelligent learning and control of autonomous robotic agents operating in unstructured environments.* (Information Sciences: Volume 145, Issue 1-2: 2002), 1-12.
[13] H.R. Everett. Sensors for Mobile Robots: Theory and Application. Wellesley, Mass.: A.K. Peters, 1995.

Appendix A

Summary Paper

Proceedings of The National Conference
On Undergraduate Research (NCUR) 2005
Virginia Military Institute
Washington and Lee University
Lexington, Virginia
April 21 - 23, 2005

Self-Adjusting Finite State Machines: An approach to Real-Time Autonomous Behavior in Robots

Scott Schwartz
Department of Computer Science
Trinity University
One Trinity Place
San Antonio, Texas
78212-7200. USA

Faculty Advisor: Dr. Maurice Eggen

Abstract

In the Robotics industry, it is a frequent requirement that robots operate in real-time. The usual approach to this issue involves creating robots driven entirely by direct environmental input rather than complicated planning and decision-making AI. This approach means that the current state of the robot in relation to its environment exclusively determines the actions of the robot. In the simplest terms, this approach creates a Finite State Machine (FSM). Clearly, a standard FSM is completely pre-deterministic upon its creation. This is a drawback which immediately disallows the robot to cope with dynamic environments in an autonomous manner. This research suggests a solution to this problem, while still maintaining real-time performance of the FSM structure, through the development of a Self-Adjusting FSM (SA-FSM). A SA-FSM is a FSM with an additional module which adds, removes, and adjusts specific states of its FSM structure. By adjusting its FSM the SA-FSM will have the basis for autonomous attributes. It will be capable of coping with drastic changes in its environment by making necessary fundamental adjustments to its behavior. Through this mechanism, the process of learning can be implemented. In this regard, only the inherent learning/inference algorithms the SA-FSM employs to adjust its FSM determine the complexity of the behavior produced by a SA-FSM based robot.
Keywords: Robots, Autonomous Robots, AI, Real-Time, FSM.

1. Introduction

An Autonomous Mobile Robot (AMR) is a robot that independently navigates and operates in its environment in order to perform its objectives. An AMR may incorporate learning and adaptation in order to continually adjust its behaviors so that it appropriately interacts with the environment in order to accomplishing its tasks (1).

The ideal AMR would be a robot which behaved in a way reminiscent of the earth's organic inhabitants. These creatures are able to cope in this unstructured situation through adaptation. They are equipped with all the necessary components to cope with their environment through learning.

1.1. motivation

An AMR can perform tasks in dangerous environments where humans are unable to perform the tasks (2). An AMR can be used in areas where, for whatever other reasons, humans cannot go (1). An AMR can also be used for more mundane tasks that are too dirty or too dull for humans, such as cleaning, inspection, and surveillance (3). An AMR would be useful for many reasons. Consider the following example. There are currently no robots operating on Mars that are not dependent on the continual support of human instruction. If there were and AMR for this task, it could act independently to pursue its objectives without the risk of

human controller error, slow communication with the controller, or even loss of communication with the controller. All these risks compromise the usefulness of the robot. This illustrates that if an autonomous real-time AI was developed, the way robots are used would be revolutionized. It appears that all the necessary pieces are in place for the creation of an AMR to be accomplished. But this advance has not yet been made, as the lack of such an entity verifies.

1.2. requirements

For an AMR to be useful and effective there are three criteria it must meet. First, it must be *intelligent*. That is, its decision making with respect to its actions must not be suspect. It must be efficient in the manner in which it accomplishes its goals. Second, it must be *autonomous*. It must allow change and adaptation in its behaviors in order to maintain its functionality under varying environments. It must be a completely self-contained mechanism that was capable of making fundamental adjustments to its own nature in order to sustain its goals. Finally, it must capably operate in a real-time fashion. If it does not act in time with its environment, it may impractical for use in many important situations. If it did not truly act in real-time, it could compromise itself by not responding to a danger in its environment in time to escape it. The DIA will be a mechanism which must meet "hard" deadlines.

1.3. standard approaches

The mechanical hardware that is required for constructing a more than satisfactory vehicle for AMR purposes is already in place. The problem now lies in the development of a controlling mechanism which realizes the behaviors required for an AMR. The project of producing such a controller has thus far been approached from two different general paradigms.

The first of these strategies is the *Deliberative* approach. This approach is also known as the *Traditional*, *Functional*, and *Symbolic* approach. The methodology underlying this approach is the sense-think-act cycle (1). The approach says that the robot should proceed through the following series of steps (3). First, the robot's sensors record the input. Then, any computation required to format the sensors input is done. Once the sensors input is in a usable form, it is all mixed together to represent the current state of the environment for the robot. This model representation of the environment is then used by the robot in order to decide how it should proceed. Once the plan is developed, the robot will execute the plan. This cycle will then be repeated. A problem with this approach is that it requires correct execution at every step. This type of system will not be very robust (1). This does not reflect positively on AMR goals. Also, this approach requires an intensive computational process which may create a bottleneck which adversely affects the environmental sampling rate (1). This same problem may also slow down the reaction time of the mobile robot (1). This could be particularly disastrous in an AMR application. However, this approach is not completely unappealing, because a deliberative approach to problem solving would seem to have some merit as a concept in an AMR application.

The second of these strategies is the *Reactive* approach. This approach is based upon Rodney Brooks' *Subsumption Architecture* proposal (4). It is also called the *Sub-Symbolic* approach. In this paradigm, the environment is used as its own best model. The robot simply reacts to occurrences in the environment, so the instructions for the robot's actions and behavior are directly from the environment around the robot (4). The behaviors of the robot all run in parallel waiting to be triggered (4). Once a behavior is triggered, its commands are fired and the resulting action adjusts the robot's position relative to the environment, possibly triggering additional behaviors. The individual behaviors are not meant to be complex, but by combining and layering a number of reactive behaviors, an advanced and complex intelligence begins to emerge (4). This approach has already met with impressive success as insect-like intelligence has already been demonstrated through the use of this methodology (4). The disadvantage is that this approach does not appear to provide an easy way to allow developing and reasoning about a plan, as the *Deliberative* approach does (1). This attribute may be an important attribute in an AMR. However, the advantages that this approach offers to AMR applications are extensive. This approach is simple and doesn't require a megalithic hierarchical programming achievement (1). Because it is based on simple behaviors it is easy to extend. Adding new behaviors does not require in massive adjustments to the current system (1). It supports multiple parallel goals through independent individual concurrent behaviors (1). It is very robust (1). If one component behavior has been lost, this fact need not effect the execution of the

other behaviors. There is no computation to create a time bottleneck on the system. The robot simply receives input which it is designed to react to.

Both of these approaches have strengths and weaknesses with respect to the AMR objectives. Therefore, there is often some sort of combination of the two methodologies which attempts to extract the best of each idea, and remove the negatives. These attempts employ a *Hybrid Deliberative/Reactive* paradigm. Such an approach is advisable in the AMR project.

2. Concept

The approach suggested here to produce an AMR which meets the given requirements entails the development of a *Self-Adjusting Finite State Machine (SA-FSM).* The primary idea behind this concept is to extend the *reaction* paradigm in the following way: Instead of endowing the robot with a fixed set of *reaction behavior* attributes upon creation, the robot is allowed to create, adjust, and remove the specific *reaction behavior* attributes while it is operation. This system will be based on continuous background processing which applies *deliberative* methods which perform the necessary adjustments to the FSM. Additionally, this underlying foundation will also be able to influence other matters of the robots operation. This suggestion will provide the necessary immediate interaction capabilities, while also providing a structure for advanced behavior capabilities. Additionally, an FSM encompasses all forms of computation, so this approach appears to be well founded, at least at this initial outset.

3. Methodology

This section presents some very broad suggestions which can be useful for developing an intelligent autonomous real-time mobile robot based on the approach put forward above.

3.1. component distribution

When designing an AMR it would be constructive to maximize the usage of computing resources, so long as it does not overcomplicate the system. The robot should not be limited to a single processor with limited memory which cannot adequately meet the AMR requirements. A good design for an AMR controlling architecture will emphasize a distribution of independent functionalities which can be organized in such a way as to meet the requirements through concurrency and parallelization. A Distributed Intelligent Agent (DIA) entity is a task-oriented entity whose functionality is distributed among individual intelligent agents (5). Each of these agents will have its own processing power to carry out their functional actions based on inputs, a knowledge-base/inference-engine, and general objectives. A DIA system is divided up into components which behave and interact together in order to accomplish the overall objectives of the system.

3.2. modular interface

Using the DIA architecture immediately provides for a modular view of the entire system. By breaking apart sub-functionalities and distributing them to independent processing agents, the menial details of each agent can be hidden from the others. Instead, the agents only know the *high level* specifications of any other agent. This simplifies the development of individual agent units, because they can now interact with each other on a higher level. This offers a simplifying extension for the control of the AMR. The control will take a *high level* approach. It will not deal with the very low level details of its any of its actions, such as movement. Instead, the commands the robot issues will be *high level modular commands* that do not involve step by step execution instructions, such as “move forward.” As far as the robot is concerned, these actions are very simple, while in actual fact they can be as complicated as desired. The commands which are dealt with will be non-atomic commands that the mechanical parts carry without exposing all the involved complications that the actual execution entails. The commands the robot wishes to execute will be sent to independent agent units which execute the instructions self-sufficiently. The actions are completely the responsibility of a separate processing unit.

3.3. intrinsic AMR character

An AMR is built for a specific purpose. Its designers equip the robot with all the sensors and actuators it will require for its undertaking. Without this, the robot will not be able to behave effectively with respect to its goals. The goals themselves are determined and equated with positive sensations for the robot. Also, the weaknesses of the robot are determined and equated as negative sensation for the robot.

The intrinsic behavior characteristics of the robot are formed directly from the "satisfactions" and "pains" related to the robot's goals and weaknesses. These are components of the robot's nature. They are not learned or adjusted after the design stage, and are instead the behaviors that the robot automatically submits to. Just as humans do not learn what the sensations from tickling or extreme heat, these *laws* are the built in attributes of the robot. These intrinsic traits of the AMR are its *laws*. The AMR can base its behaviors on its intrinsic *laws.*

4. Controller Considerations

The *Hybrid Deliberative/Reactive* approach will be employed as the underlying foundation of AMR control. The *deliberative* component will be considered as an overarching entity which continually presides over the proceedings of the robot. In this entity, notions of *preference, judgment,* and *will* can be made. *Suggestions* as to the behavior of the robot are also seen to be generated in this entity. The *reactive* component will be the SA-FSM that was described earlier. This will provide the necessary real-time interaction with the environment through actions which are triggered by environmental stimuli. The advancement to *Hybrid Deliberative/Reactive* paradigm is to allow the inputs which trigger responses be added, adjusted, and removed as the robot sees fit through learning. In this way, the *behavioral* approach can be followed while the *deliberative* approach can be used concurrently.

4.1. Reactive SA-FSM

4.1.1 states

At any given time, the current status of all the sensors indicates the *state* that the robot is in. Sensors can be divided into two general classifications based upon the measurement of the sensor as either *output* or *input*. The output sensors measure the effects of robot actions on the environment. The key with this category is that these sensor readings are generated from changes in the environment which result directly from the robot actions. Thus, they are *output*. The *input* sensors measure things that are attributes of the robot's environment. These may be measurements that indicate the status of a certain piece of the robot. But these are not changes resulting from actions taken by the robot, and so are still categorized as *input* sensors.

4.1.2 events and critical events

Certain *states* have particular implications for the robot's *laws*. A *critical event* is a *state* directly related to a *law*. The nature of the *law* reflects the reaction which must occur in this *state*. A *state* which is related to *critical event*, but is not the *critical event* is called an *event.* If the *critical event* is negative, then the related *event* gives information concerning how to avoid it. If the *critical event* is positive, the information will concern how to achieve it. Sensor *states* related to the robot's *laws* are the stimuli for immediate SA-FSM reactions.

An *event* can be classified into three categories. There are *environmental events*. These involve *states* related to the environment such as water, gravel, slope, gravel, or anything else that can be come up with. There are *entity events* which are *states* involving other agents in the environment that are dynamic to the robot in that it can interact with them. And there are *preservation events*. These are *state* condition necessary for the robot's operation such as heat, radiation, and pressure.

4.1.3 rules

The *rules* correspond to the *event* and *critical event states. Rules* are reaction behavior that is immediately activated by a SA-FSM reaction. They are the way the robot should act in a given *event state.* If the robot

reaches an *event state*, it will know how to best respond through the SA-FSM and how it has been managed by the *deliberative* portion of the controller. The behavior, or *rule*, is fired for the *event state*. Each special state may have several *rules* attached to it. These specify the actions that are or are not desirable in the given state.

4.2. Deliberative SA-FSM Interaction

Just as the robot is given the correct mechanical actuators for interaction with its environment and is given the correct sensor array to allow it accomplish its task, the AMR must be endowed with all the necessary learning algorithms. These will be run continuously in the *deliberative* component of the controller. Initially, the robot is aware only of its *laws*. It must generate *rules* for those *laws*. It must then learn new related *events*, and the corresponding *rules*, in order to incorporate them into the SA-FSM. Also, the learning must be done before any real consequences can be met. The robot cannot learn from being destroyed.

The AMR needs to learn about its *environment*. Certain sensor *states* related to the environment will be *events* which will trigger necessary responses. The AMR also needs to learn about *itself*. If it knows what its actions do, it can base all its *rules* on its knowledge of its actions. In this way, the AMR will always perform the best action in the given current *event*. To use this approach, the *deliberative* learning algorithms must continually monitor the results of robot's actions so that if an actions behaves differently than it once did, this change can be accounted for, and the correct adjustments be made.

The algorithms will need to learn in spite of sensor inaccuracy. They will need to learn when the sensors can be used as predictors of an *event*. They will also need to learn when the sensors do not predict anything even though a change has occurred and must be learned. This case may require a context switch to allow the robot to perform adequately under both scenarios.

Changes to the system need to be made in all the relevant places. This must be done in unison and in parallel. It must be a coordinated synchronous manner is imperative. Part of the system cannot be changed while the other is unaware of the change. If this were to happen, the two components would be unable to communicate to one another. The adjustments made to the SA-FSM need to be done quickly and quietly, behind the scene in order to meet all the AMR requirements.

5. Summary

This paper suggests a new approach to AMR control. This approach builds on the best aspects of the current practices in order to maintain the benefits they provide, while establishing a new methodology for AMR which provides additional capabilities. The proposed methodology is the SA-FSM paradigm. This methodology allows for an advanced *reactive* approach which is not subject to initial designated behavior constraints because the behaviors can be adjusted during AMR operation.

6. References

Books

1. Ulrich Nehmzow. *Mobile Robotics: a practical introduction*, 2nd ed. (New York: Springer, 2003), 8-9, 11, 14-17, 20-21.
2. Peter Menzel and Faith D'Aluisio. *Robo sapiens: evolution of a new species*. (Cambridge, Mass.: MIT Press, 2000), 138-141.
3. Joseph L. Jones, Bruce A. Seiger, and Anita M. Flynn. *Mobile robots: inspiration to implementation*, 2nd ed. (Natick, Mass.: A.K. Peters, 1999), 337.

Journals (Print)

4. Rodney A. Brooks, *Intelligence without representation**. (Artificial Intelligence Volume 47, Issue 1-3: 1991), 139-159.
5. Maurice Eggen and Gerald Pitts, *Distributed Intelligent Agents: A New Approach to Distributed Artificial Intelligence in Robotic Systems*. (PDPTA: 2003), 589-595.

Appendix B

Project Model

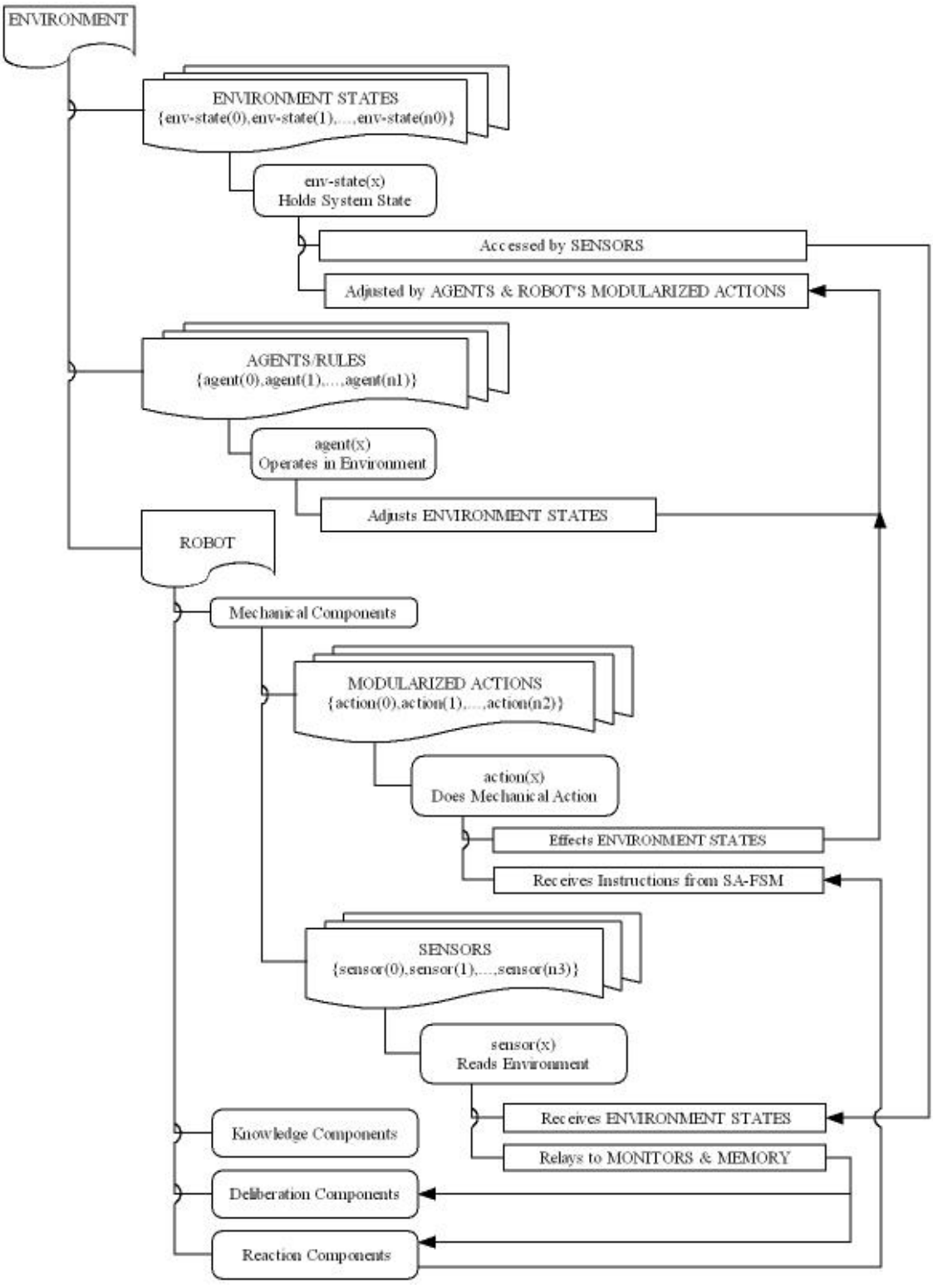

ENVIRONMENT
ENVIRONMENT STATES
{env-state(0),env-state(1),...,env-state(n0)}
env-state(x)
Holds System State
Accessed by SENSORS
Adjusted by AGENTS & ROBOT'S MODULARIZED ACTIONS
AGENTS/RULES
{agent(0),agent(1),...,agent(n1)}
agent(x)
Operates in Environment
Adjusts ENVIRONMENT STATES
ROBOT
Mechanical Components
MODULARIZED ACTIONS
{action(0),action(1),...,action(n2)}
action(x)
Does Mechanical Action
Effects ENVIRONMENT STATES
Receives Instructions from SA-FSM
SENSORS
{sensor(0),sensor(1),...,sensor(n3)}
sensor(x)
Reads Environment
Receives ENVIRONMENT STATES
Relays to MONITORS & MEMORY
Knowledge Components
Deliberation Components
Reaction Components

Figure 1

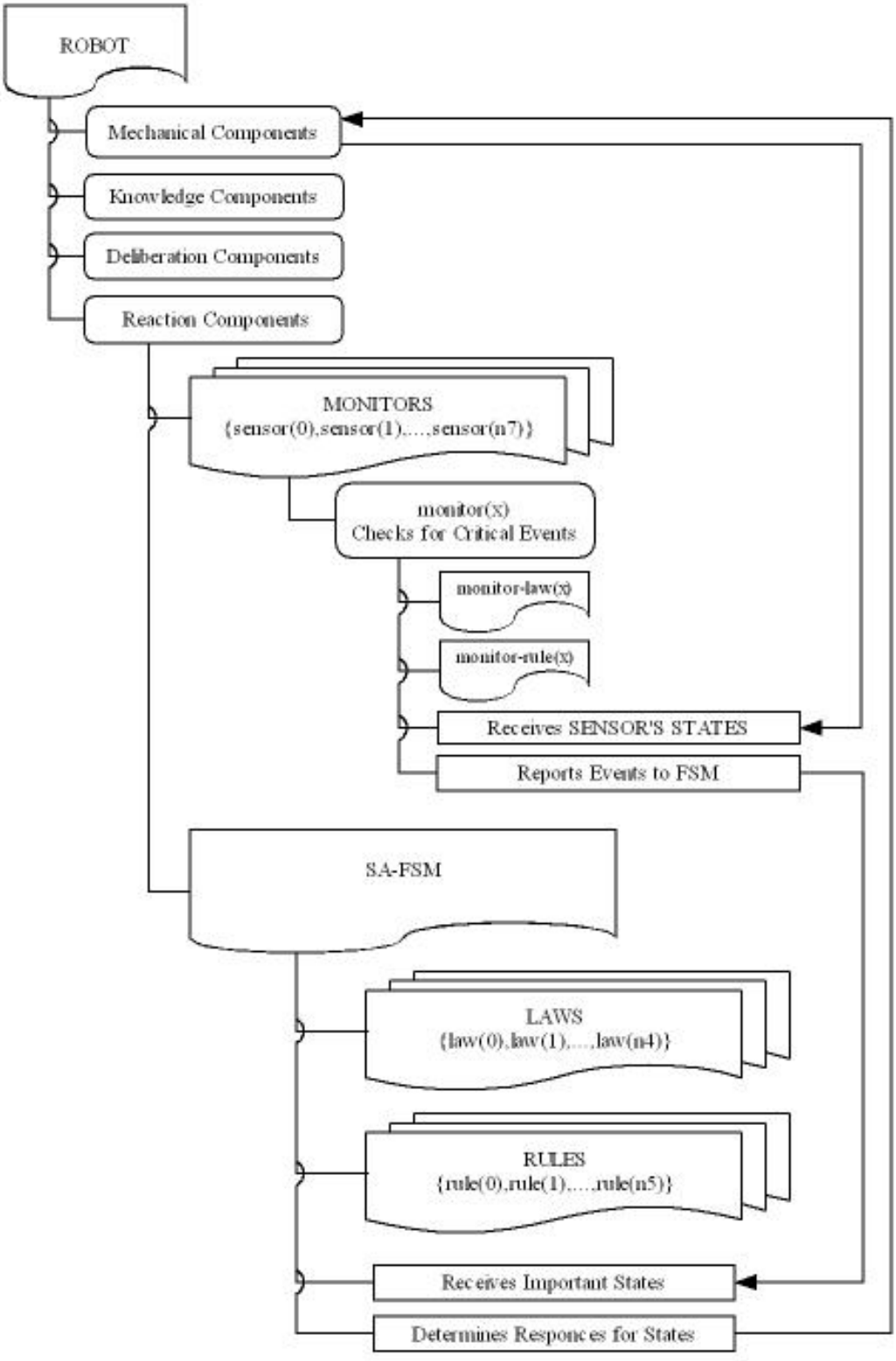

ROBOT
Mechanical Components
Knowledge Components
Deliberation Components
Reaction Components
MONITORS
{sensor(0),sensor(1),...,sensor(n7)}
monitor(x)
Checks for Critical Events
monitor-law(x)
monitor-rule(x)
Receives SENSOR'S STATES
Reports Events to FSM
SA-FSM
LAWS
{law(0),law(1),...,law(n4)}
RULES
{rule(0),rule(1),...,rule(n5)}
Receives Important States
Determines Responces for States

Figure 2

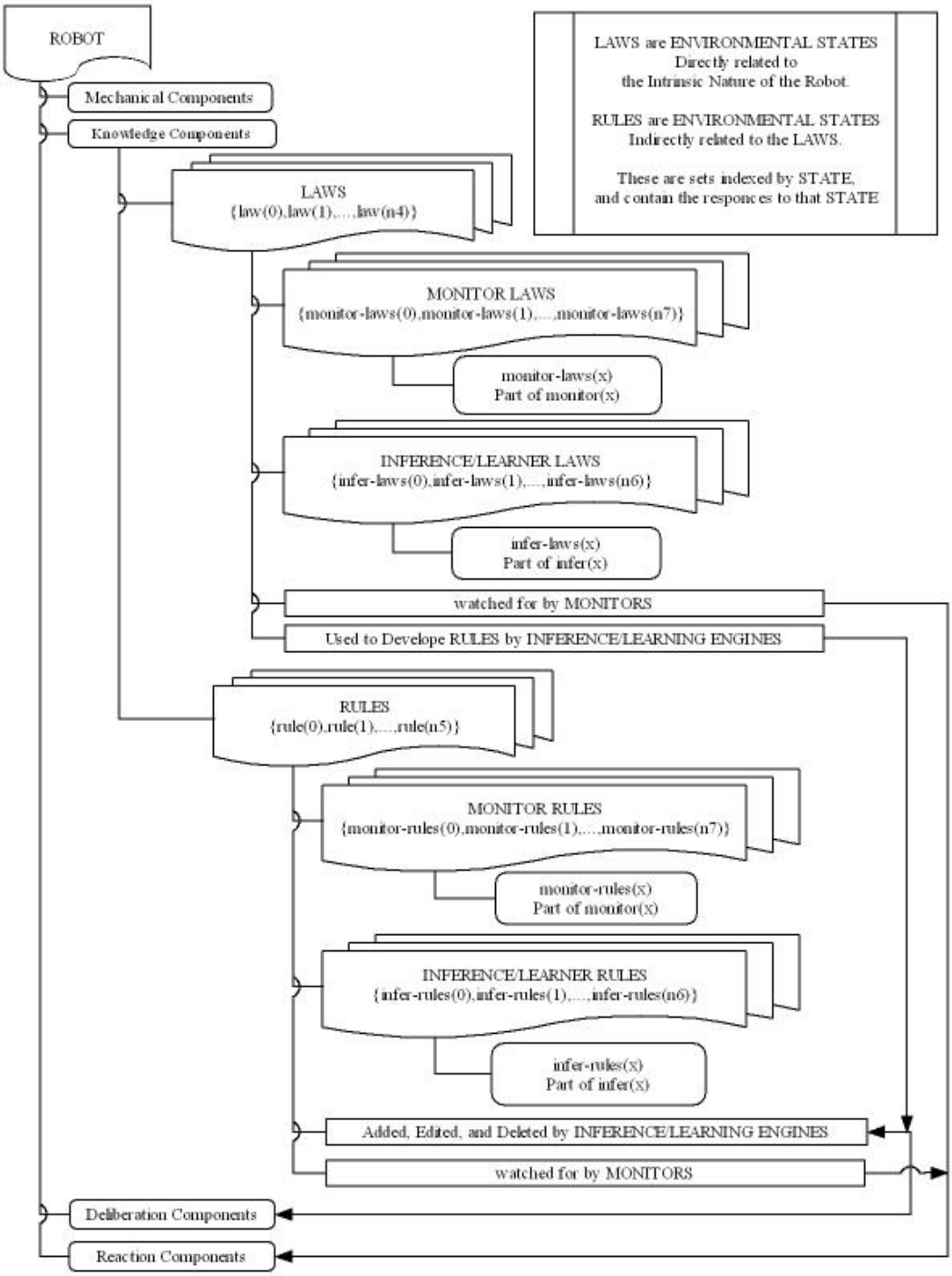

ROBOT
Mechanical Components
Knowledge Components
LAWS
{law(0),law(1),....,law(n4)}
LAWS are ENVIRONMENTAL STATES
Directly related to
the Intrinsic Nature of the Robot.
RULES are ENVIRONMENTAL STATES
Indirectly related to the LAWS.
These are sets indexed by STATE,
and contain the responces to that STATE
MONITOR LAWS
{monitor-laws(0),monitor-laws(1),....,monitor-laws(n7)}
monitor-laws(x)
Part of monitor(x)
INFERENCE/LEARNER LAWS
{infer-laws(0),infer-laws(1),....,infer-laws(n6)}
infer-laws(x)
Part of infer(x)
watched for by MONITORS
Used to Develope RULES by INFERENCE/LEARNING ENGINES
RULES
{rule(0),rule(1),....,rule(n5)}
MONITOR RULES
{monitor-rules(0),monitor-rules(1),....,monitor-rules(n7)}
monitor-rules(x)
Part of monitor(x)
INFERENCE/LEARNER RULES
{infer-rules(0),infer-rules(1),....,infer-rules(n6)}
infer-rules(x)
Part of infer(x)
Added, Edited, and Deleted by INFERENCE/LEARNING ENGINES
watched for by MONITORS
Deliberation Components
Reaction Components

Figure 3

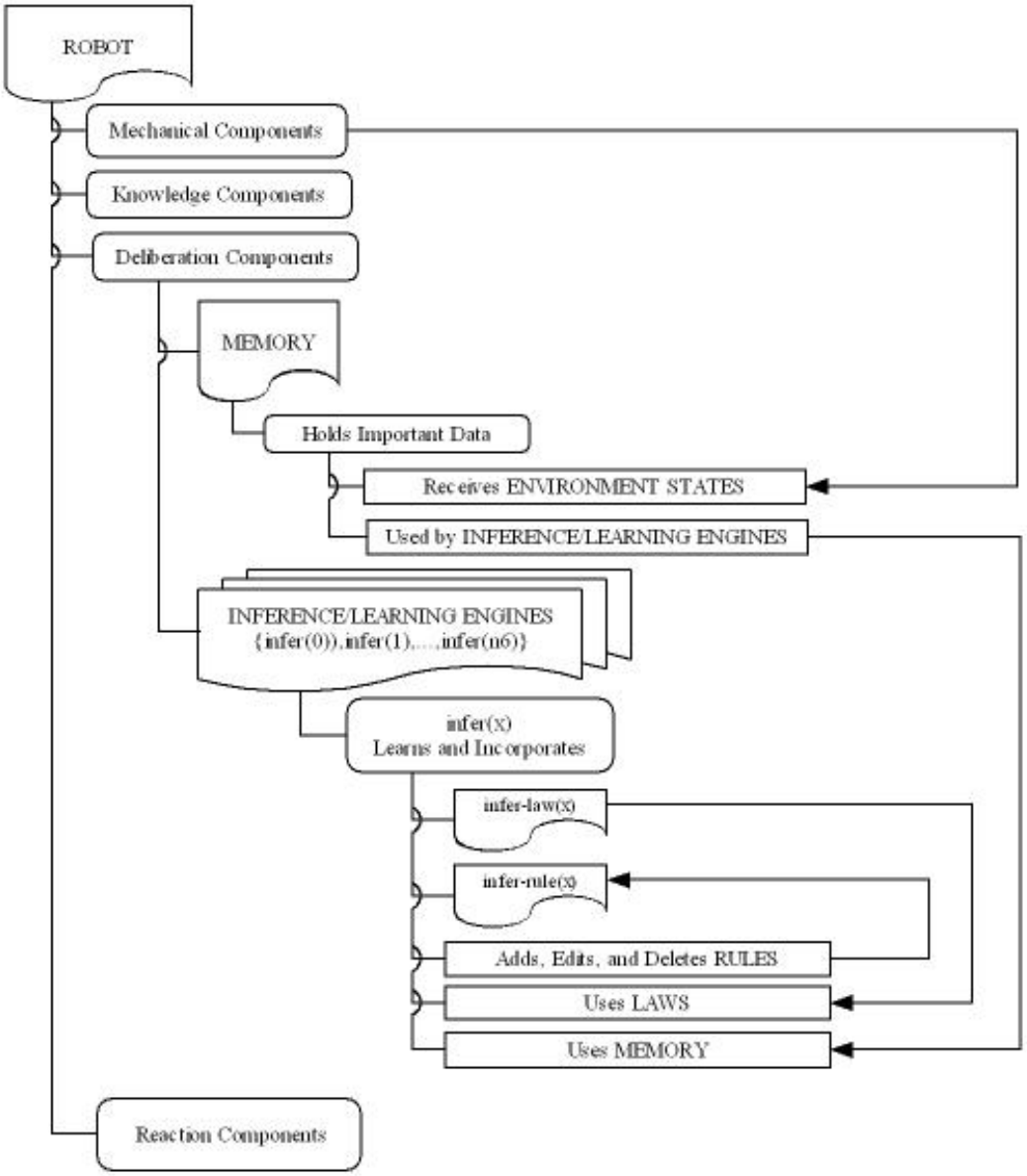

ROBOT
Mechanical Components
Knowledge Components
Deliberation Components
MEMORY
Holds Important Data
Receives ENVIRONMENT STATES
Used by INFERENCE/LEARNING ENGINES
INFERENCE/LEARNING ENGINES
{infer(0),infer(1),....infer(n6)}
infer(x)
Learns and Incorporates
infer-law(x)
infer-rule(x)
Adds, Edits, and Deletes RULES
Uses LAWS
Uses MEMORY
Reaction Components

Figure 4

Appendix C

Implementation Source Code

```
//  ENVIRONMENT.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  The Environment
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// this is the base class.  Everything works off this.
// it manages the state of the world that is operated in.
// this file houses all the states of the environment
// the environment manages it's state
//
// it also houses the robot, and agents, and whatever
//
// the environment recieves the actions of the robot
// and the actions of agents acting in it
// and adjusts its state as it is affected
// the environment may be very complicated and
// may manage many other things happening
//
// drastic changes to the behavior of the robot can be similated.
// broken tire, moving on ice...just adjust the result of actions
//
// environment communicates with sensors,
// gets communicated too by actions
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

import java.io.*;
import java.util.*;
import java.net.*;

import robot.*;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  ENVIRONMENT
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// this is the base class.  Everything works off this.
// and it manages the state of the world that is operated in.
//
// the data here holds/is
// states of the environment and related functionality
//
// main is the world running
//
// here, the environment will have a set of states that
// are important to the robot.
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public class ENVIRONMENT{
  public double power_level;

  //initialization and setup of environment
  public ENVIRONMENT() throws Exception{
    power_level = 5;
```

```
  }

  //initialization and setup of environment
  public void applyRobotActions(int action) throws Exception{
    if(action == 0)
      power_level -= 1;
    if(action == 1)
      power_level -= 2;
    if(action == 2)
      power_level += 1;
  }

  public static void main(String args[]) throws Exception{
    ENVIRONMENT world = new ENVIRONMENT();
    ROBOT i = new ROBOT();
    int j;

    //initial reading
    i.power_sensor.takeReading(world.power_level);
    //sensor reading passed on
    i.memory.getInput(i.power_sensor.relayReading());
    i.power_monitor.getReading(i.power_sensor.relayReading());

    //robot loop
    for(j=0;j<1000;j+=1){
i.power_events.print();

    //state determined
    i.power_monitor.getState(
      i.power_events.readingToState(i.power_monitor.relayReading()));

    //state sent to fsm, and reaction determined
    i.fsm.receiveState(i.power_monitor.relayState());
    i.fsm.determineReaction(
      i.power_events.stateToAction(i.fsm.relayState()));

    //action fired, and results applied
    i.power_action.receiveInstruction(i.fsm.fireReaction());
    i.memory.getOutput(i.power_action.performAction());
    world.applyRobotActions(i.power_action.performAction());

    //results read by robot
    i.power_sensor.takeReading(world.power_level);
    //sensor reading passed on
    i.memory.getInput(i.power_sensor.relayReading());
    i.power_monitor.getReading(i.power_sensor.relayReading());

    //learning based on event
    i.power_events.updateEvents(
      i.power_action.performAction(),
      i.memory.actionQuery(i.power_action.performAction()));
```

```
    //cycle restarted
System.out.print("\n" + j + "\n");
    }
  }
}
```

```java
//  ROBOT.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><>
//  The Robot
//<><><><><><><><><><><><><><><><><><><><>
// the robot houses the mechanical interaction with the environment
// the sensors are housed in the robot
// the actions the robot can perform are contained as part of the robot.
// it's just a house to group everything in.
//
// the robot also houses its ai stuff
//
// There's a important distinction between sensors:
// INPUT vs. OUTPUT, though they will be treated very similarly
// these represent different cycles
//
//<><><><><><><><><><><><><><><><><><><><>

package robot;

import java.io.*;
import java.util.*;
import java.net.*;

import robot.mechanical.SENSORS.*;
import robot.fsm.MONITORS.*;
import robot.fsm.*;
import robot.mechanical.ACTIONS.*;
import robot.ai.*;
import robot.ai.LEARNERS.*;

//<><><><><><><><><><><><><><><><><><><><>
//  ROBOT
//<><><><><><><><><><><><><><><><><><><><>
// sensors and related functionality
// actions and related functionality
// ahhh....this is a nice way to group actions!!!
// turning on the robot, all sensors are started
// all mechanical interactors inititiated
// initialize parts of the robot
//<><><><><><><><><><><><><><><><><><><><>
public class ROBOT{
  //<><><><><><><><><><><><><><><><><><><><>
  //  data
  //<><><><><><><><><><><><><><><><><><><><>
  // power modules
  // memory module
  //<><><><><><><><><><><><><><><><><><><><>
  public POWER_SENSOR power_sensor;
  public POWER_SENSOR_MONITOR power_monitor;
  public POWER_ACTION power_action;
```

```
  public MEMORY memory;
  public POWER_EVENTS power_events;
  public FSM fsm;

  public ROBOT() throws Exception{
    power_sensor = new POWER_SENSOR();
    power_monitor = new POWER_SENSOR_MONITOR();
    power_action = new POWER_ACTION();
    memory = new MEMORY();
    power_events = new POWER_EVENTS();
    fsm = new FSM();
  }

/*
//this would preferably be a command that just tells
//all the sensors to fire
//then the sensors just take over and pass on the info
//just like they're supposed to.
  public void gather(double reading) throws Exception{
    power_sensor.takeReading(reading);
    power_monitor.getReading(power_sensor.relayReading());
    memory.getInput(power_sensor.relayReading());
  }

//this would preferably be a command that just tells
//all the actions to fire
//or maybe reads in all the actions and syncronizes them or something
  public double respond() throws Exception{
    power_action.receiveInstruction(2);
    return power_action.performAction();
  }
*/
  public static void main(String args[]) throws Exception{
  }
}
```

```
//  POWER_ACTION.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  ACTIONS
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// completely self contained modular unit
// that generates a change in the environment!
//
// receives instructions from FSM just before the
// previous instruction is finished and continues with
// new instruction.
//
// different cycles depend on action
// some cycles need to happen faster...
// others slower...
// :::The FSM continually gives commands, but
// the actions actually carry out the commands
// they've got like a buffer, to see if the command
// is actually already being done or something
//
// yes, they just wait for a new command, they
// continue what they're doing...
// now, this will get complicated as more complicated
// actions are allowed, like varying speeds and power...
//
// recieve instruction on what to do given state from AI
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

package robot.mechanical.ACTIONS;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  POWER_ACTION
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// This is a power action,
// it does something that effects power level of robot
//
// there is a list of possible actions related to this
// modular action...this is a dependent set of actions...
// one or the other is performed
//
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public class POWER_ACTION{

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  data
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // the current power action that is being performed at any time
```

```
  // 0 is idle, -1 on power
  // 1 is working, -2 on power
  // 2 is resting, +1 on power
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  private int action;

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  POWER_SENSOR()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // initialization of power action
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public POWER_ACTION() throws Exception{
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  void receivedInstruction(int instruction)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // get instruction from FSM
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public void receiveInstruction(int instruction) throws Exception{
    action = instruction;
    System.out.print("[POWER_ACTION] receiveInstruction: " + action + "\n");
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  void receivedInstruction(int instruction)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // do instruction in environment
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public int performAction() throws Exception{
    System.out.print("[POWER_ACTION] performAction: " + action + "\n");
    return action;
  }

  public static void main(String args[]) throws Exception{
  }
}
```

```
//  POWER_SENSOR.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><>
//  SENSORS
//<><><><><><><><><><><><><><><><><><><><>
// The Sensor classes are
// abstractions or simulations of real world sensors.
//
// Noise and general realisticness of robot sensors adjusted
// in individual sensor classes
//
// There will be many sensors on a given robot
// the sensors are part of the robot which
// resides in the environment
//
// In our implementation for robot these sensors relay their
// readings directly to their Monitors and memory.
// Monitors check the measurements
//<><><><><><><><><><><><><><><><><><><><>

package robot.mechanical.SENSORS;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><>
//  POWER_SENSOR
//<><><><><><><><><><><><><><><><><><><><>
// This sensor measures the overall power level of the robot
//
// To read the sensor, you run a call to environment
//<><><><><><><><><><><><><><><><><><><><>
public class POWER_SENSOR{

  //<><><><><><><><><><><><><><><><><><><><>
  //  data
  //<><><><><><><><><><><><><><><><><><><><>
  // power sensor reading at any given time
  //<><><><><><><><><><><><><><><><><><><><>
  private double power_reading;

  //<><><><><><><><><><><><><><><><><><><><>
  //  POWER_SENSOR()
  //<><><><><><><><><><><><><><><><><><><><>
  // initialization of power sensor reading
  //<><><><><><><><><><><><><><><><><><><><>
  public POWER_SENSOR() throws Exception{
    power_reading = 0;
  }
```

```
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  void takeReading(double reading)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // This function is called and passed the actual value
  // that the sensor is trying to read.
  //
  // Any noise that we would like to simulate will be
  // applied and then the
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public void takeReading(double reading) throws Exception{
    power_reading = sensorNoise(reading);
    System.out.print("[POWER_SENSOR] takeReading: " + power_reading + "\n");
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  double sensorNoise(double reading)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // Any noise that we would like to simulate will be
  // done so in this function
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public double sensorNoise(double reading) throws Exception{
    System.out.print("[POWER_SENSOR] sensorNoise: " + reading + "\n");
    return reading;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  double relayReading()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // this function is called and returns the value
  // of the sensor reading
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public double relayReading() throws Exception{
    System.out.print("[POWER_SENSOR] relayReading: " + power_reading + "\n");
    return power_reading;
  }

  public static void main(String args[]) throws Exception{
  }
}
```

```
//  POWER_SENSOR_MONITOR.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  MONITORS
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// The Monitor classes monitor the the sensors on a given robot.
//
// In our implementation for robot these sensors relay their
// readings directly to their Monitors, who check the measurements
// to determinie the state they're in
//
// these classes deal directly with the sensors,
// they receive the sensor data and check it for
// an event (could events span multiple sensor receptions?)
//
// each monitor has a set of
// laws: intrinsic to the robot and its sensors
// states: special states that are related to the laws
// in some way.  the states help us do somthing
// related to the laws.
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

package robot.fsm.MONITORS;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  POWER_SENSOR_MONITOR
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// This monitors the POWER_SENSOR
//
// when it receives an input from the sensor
// it checks if it is a special state and if so
// it generates an event...
//
// this event is sent to the FSM who determines
// the reaction to its current state
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public class POWER_SENSOR_MONITOR{

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  data
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // received power sensor reading at any given time
  // state of robot given monitor...
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  private double power_reading;
  private int power_state;
```

```
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  POWER_SENSOR_MONITOR()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// initialization of power sensor monitor state
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public POWER_SENSOR_MONITOR() throws Exception{
  power_state = 0;
}

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  void getReading(double reading)
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// reading received from sensor
// the reading is then checked
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public void getReading(double reading) throws Exception{
  power_reading = reading;
  System.out.print("[POWER_SENSOR_MONITOR]  getReading:  "  +  power_reading  +
"\n");
}

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//   void getState()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// checks the just received reading from sensor
// to see if the state is an event
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public void getState(int state) throws Exception{
  power_state = state;
  System.out.print("[POWER_SENSOR_MONITOR] getState: " + power_state + "\n");
}

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//   int relayState()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// tells FSM the state
// so it can react
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public int relayState() throws Exception{
  System.out.print("[POWER_SENSOR_MONITOR] relayState: " + power_state + "\n");
  return power_state;
}

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//   int relayReading()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// used to determine state
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public double relayReading() throws Exception{
  return power_reading;
}
```

```
  public static void main(String args[]) throws Exception{
  }
}
```

```
//  FSM.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  FSM
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// laws...the "pain" or "pleasure" inherrent in sensors
// this is a constant...
// what is the form of such a thing?
// these intersecting of these is a critical event
// this generates a sample for learning
//
// events...sensor readings which predict above...general, over all...
// this is what is learned
// what is the form of such a thing?
// these are simply changes in state which we have an understinding
// of what they mean, so we have rules for what to do with this
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

package robot.fsm;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  FSM
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// recieves state
// knows responce
// generates responce
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public class FSM{
  int power_state;
  int power_action;

  public FSM() throws Exception{
    power_action = 0;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  receiveState()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // from monitors
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public void receiveState(int state) throws Exception{
    System.out.print("[FSM] receiveState: " + state + "\n");
    power_state = state;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  determineReaction()
```

```
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // given the state, we react a way fast
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public void determineReaction(int action) throws Exception{
    System.out.print("[FSM] determineReaction: " + action + "\n");
    power_action = action;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  relayState()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // helps for determineReaction
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public int relayState() throws Exception{
    return power_state;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  fireReaction()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // send command to actions
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public int fireReaction() throws Exception{
    System.out.print("[FSM] fireReaction: " + power_action + "\n");
    return power_action;
  }

  public static void main(String args[]) throws Exception{
  }
}
```

```
//  MEMORY.java
//  scott schwartz

//<><><><><><><><><><><><><><><><><><><><><><><>
//  The Memory
//<><><><><><><><><><><><><><><><><><><><><><><>
// This stores all the information from the sensors
// the place where samples are kept
//<><><><><><><><><><><><><><><><><><><><><><><>

package robot.ai;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><><><><>
//  MEMORY
//<><><><><><><><><><><><><><><><><><><><><><><>
// General information on the workings
//<><><><><><><><><><><><><><><><><><><><><><><>
public class MEMORY{

 //<><><><><><><><><><><><><><><><><><><><><><><>
 //  data
 //<><><><><><><><><><><><><><><><><><><><><><><>
 // the different history that is kept here
 //<><><><><><><><><><><><><><><><><><><><><><><>
 double [] sensor;
 int s_count;
 int [] action;
 int a_count;

 //<><><><><><><><><><><><><><><><><><><><><><><>
 //  MEMORY()
 //<><><><><><><><><><><><><><><><><><><><><><><>
 // initialization of the memory
 //<><><><><><><><><><><><><><><><><><><><><><><>
 public MEMORY() throws Exception{
  sensor = new double [1001];
  s_count = 0;
  action = new int [1001];
  a_count = 0;
 }

 //<><><><><><><><><><><><><><><><><><><><><><><>
 //  void getInput(double reading)
 //<><><><><><><><><><><><><><><><><><><><><><><>
 // this receives the sensor input
 //<><><><><><><><><><><><><><><><><><><><><><><>
  public void getInput(double in) throws Exception{
   sensor[s_count] = in;
```

```
      s_count += 1;
      System.out.print("[MEMORY] getInput: " + in + "\n");
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  void getInput(double reading)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // this receives the actions that were performed
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
    public void getOutput(int out) throws Exception{
      action[a_count] = out;
      a_count += 1;
      System.out.print("[MEMORY] getOutput: " + out + "\n");
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  void actionQuery(double reading)
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // returns actions results sample
  // asks for action type.  Determines samples for that action.
  // tells the result of that action
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
    public double actionQuery(int act) throws Exception{
      int sum = 0;
      int count = 0;
      int i;
      for(i=0;i<a_count;i+=1){
        if(action[i] == act){
          sum += sensor[i+1] - sensor[i];
          count += 1;
        }
      }

      System.out.print("[MEMORY] actionQuery: action " + act + " results in " + (sum /
count) + "\n");
      return (sum / count);
  }

  public static void main(String args[]) throws Exception{
  }
}
```

```
//scott schwartz
//POWER_EVENTS.java

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  LEARNERS
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// this is the table of all the states and the corresponding actions
// that should be taken under the states
//
// these guys build the states/reactions
//
// they do all the necessarily learning and maintenance of this stuff
//
// These have access to the MEMORY, and they use it to build there
// rules
//
// there is a group of units which need access to this
// They are simply given this access.
// These are fsm & MONITORS
//
// this is the table of events the monitor looks for
// this is the table of events the fsm looks for when
// it receives events from the monitor
//
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>

package robot.ai.LEARNERS;

import java.io.*;
import java.util.*;
import java.net.*;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  POWER_EVENTS
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// all the states and rules for the POWER_ACTION,
// POWER_MONITOR...
//
// it builds the events, and how to respond to them.
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public class POWER_EVENTS{

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  data
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // the laws
  // the states
  // the responces to the states
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  private int low_law;
  private int high_law;
  private double [] power_actions_result;//effect to environment of action
```

```
private int [] states;//partitions possible states into important sets
private int [][] states_actions;//allowable actions in a state
private int lastState;//marker of end of state list
private int nextAction;

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  POWER_EVENTS()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// initialization of laws for power sensor events/states
// we have laws 0 and 10, which are bad
// we have 3 power actions which we will learn results for
// we have a sigle state -- 0.  10 signifies end of states
// this means state 0 is range (0,10)
// in state 0, we can use all three actions...so far
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public POWER_EVENTS() throws Exception{
  low_law = 0;
  high_law = 10;

  power_actions_result = new double [3];
  power_actions_result[0] = 0;
  power_actions_result[1] = 0;
  power_actions_result[2] = 0;

  states = new int [10];
  states[0] = 0;
  states[1] = 10;
  lastState = 1;

  states_actions = new int [10][3];
  int i;
  int j;
  for(i=0;i<10;i+=1)
    for(j=0;j<3;j+=1)
      states_actions[i][j] = 1;

  nextAction = 0;//how we choose an action to use
}

//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
//  updateEvents()
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
// determine what action does, and then
// build state around that result so that future
// actions reflect good decision knowledge from
// what has become known.
//<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
public void updateEvents(int action, double result) throws Exception{
  int i;
  int j;

  //learn current result of action
```

```
   power_actions_result[action] = result;

   //solve equation to infer where action infringes on laws
   int boundary = (int) -result;

   //set for infringes on upper or lower laws
   if(boundary < 0)
     boundary += 10;

   //see if boundary exists, if so, done...
   for(i=lastState;i>=0;i-=1){
     if(boundary == states[i])
       return;
   }

   //build in this new result as a new partition, while keeping all old info
   //(an inference process)
   //where the result is respected so it doesn't break boundaries
   i = lastState;
   lastState += 1;
   while( (boundary < states[i]) && (i>=0) ){
     states[i+1] = states[i];
     states_actions[i+1][0] = states_actions[i][0];
     states_actions[i+1][1] = states_actions[i][1];
     states_actions[i+1][2] = states_actions[i][2];
     i -= 1;
   }

   //add the new state partition
   states[i+1] = boundary;
   states_actions[i][action] = 0;

   //make sure no original information was lost.
   //...that original partition is still imprinted on the new partition.
   //SPLIT THE PARTITION...ADD THE ADDTIONAL FEATURES, KEEP ORIGINAL
PARTION
   //AS IT WAS...JUST THE EXTRAS ARE ADDED
   if(boundary == 1)
     states_actions[0][1] = 0;//this is a hack...
       System.out.print("[POWER_EVENTS] updateEvents: did updating \n");
 }

 //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
 //  readingToState()
 //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
 // searches states for ranges, then returns state with the range
 //<><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><><>
 public int readingToState(double reading) throws Exception{
   int i;
   for(i=0;i<10;i+=1)
     if(states[i] >= reading){
       System.out.print("[POWER_EVENTS] readingToState: Reading "
```

```
        + reading + " is State " + (i-1) + "\n");
       return i-1;
     }
   return 0;
  }

  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  //  stateToAction()
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  // We have the state, this tells us the options
  // we pick an option
  //<><><><><><><><><><><><><><><><><><><><><><><><><><><><>
  public int stateToAction(int state) throws Exception{
    int choice;

    while(states_actions[state][nextAction] == 0)
      nextAction = (nextAction+1)%3;

    choice = nextAction;
    nextAction = (nextAction+1)%3;

    System.out.print("[POWER_EVENTS] stateToAction: State "
      + state + " executes action " + choice + "\n");

    return choice;
  }

  public void print() throws Exception{
    int i;
    int j;
    for(i=0;i<10;i+=1){
      System.out.print("\n");
      for(j=0;j<3;j+=1)
        System.out.print(states_actions[i][j] + " " );
    }
    System.out.print("\n\n");
    for(j=0;j<3;j+=1)
      System.out.print(states[j] + " " );
    System.out.print("\n\n");

  }

  public static void main(String args[]) throws Exception{
  }
}
```