Knowledge Representation and Learning for Robotic Systems

B.Tech Seminar Report

Submitted in partial fulfillment of the requirements
for the degree of

Bachelor of Technology

by

Sarvjeet Singh

Roll No: 99005029

under the guidance of

Dr. Pushpak Bhattacharyya

Department of Computer Science and Engineering
Indian Institute of Technology, Bombay
Mumbai

Contents

Abstract

This seminar report focuses on ”Knowledge Representation and Learning for Robotic Systems”. Three theories have been studied in this report namely Society of Mind, Subsumption Architecture and Reinforcement Learning. Society of Mind describes a basic framework for building intelligence. A model of building behavior-based autonomous robots is given in Subsumption Architecture. A mathematical model of unsupervised learning is presented in Reinforcement learning. I have outlined relative strengths and weaknesses of these theories and cited a possible way to remove the latter by combining the last two theories.

1 Introduction

1.1 Motivation

Understanding intelligence is man's dream for centuries. Intelligence that is built into natural systems like Humans is our best guide and motivation for understanding intelligence. We want to know how we can build this intelligence in our artificial systems. Researchers in Artificial Intelligence are interested in understanding the properties of living organisms so that they can build their own artificial systems that exhibit these properties for useful purposes. Many theories have been proposed to explain intelligence of biological systems we see all around us and build similar systems.

There are many types of artificial systems we build. On one hand are the systems that reside inside a computer. They are basically agents acting in an information domain. Generally these systems require limited knowledge of a domain. Examples of these systems are numerous helper utilities we use in our computers. They rarely interact with the physical world. These systems are relatively easy to make and offers limited insight into natural systems. On the other end of spectrum are the control systems of completely *autonomous* robotic creatures. These robots are required to interact with the real physical world. Because of the enormous complexity and uncertainty of this world, they require knowledge of broad domains in order to survive and function properly. Some of these kind of systems are built to operate in simulated worlds that approximately (and often poorly) model the real physical world. These systems serve no useful purpose other than giving chance to test and develop more theories. More insight in intelligence is gained by the systems that operate in real worlds. These robots *live* and operate in physical world and carry out some tasks which has some utility for whoever wanted the robots to exist and live in this world.

Recently there has been a shift in Artificial intelligence community towards building robots that operate in the real world instead of programs running in simulated environments. The intelligence systems, built by nature, which we take as our role models, interact with the real complex world. Sensors, actuators, power sources and intelligence are important components of such creatures. Choices in any part of the system architecture have major impacts upon other parts of the system. While we may be tempted to think that we can isolate the intelligence component and study it by itself, but, in general, it is very dangerous assumption. [12].

Complex behavior need not be result of extremely complex control system. Complex behavior may be simply reflection of complex environment. It may be observer who ascribes complexity to an organism and not necessarily its designer.

Autonomous mobile robots, equipped with sensors, actuators and intelligent control system, which are living and interacting with the real world are quite similar to the intelligence systems of the biological world. As discussed above building such systems might give us better insight into intelligence. Thus we would like to study how we can build intelligent control systems for such robots.

This seminar presents and discusses some of the important models and theories of Knowledge representation and learning for robotic systems. Some of the important desirable qualities that we are looking in systems that result using these models are discussed in next section.

1.2 Some desirable qualities of autonomous robots

From any model of building the control system of an autonomous robot, some qualities are desired [12]:

- **Convergence**: We should be able to demonstrate or prove that a robot is programmed in such a way that its external behavior indeed achieves a particular goal successfully.
- **Complexity**: The robot should be able to deal with the complexity of the real world environments. It should shift its attention on relevant sensations rather than being overwhelmed with multitudes of data.
- **Uncertainty in sensors**: The robot should take into account the inherent uncertainties and noise in sensors.
- **Coherence**: Even though many behaviors may be active at once, or are being actively switched on or off, The robot should still appear to have coherence of actions and goals to an outside observer.
- **Salience**: The behavior that are active at a given time should be salient to the situation the creature finds itself in.
- **Adequacy**: The behavior selection mechanism must operate in such a way that the long term goals that creature is supposed to achieve are met.

1.3 Report layout

Section 2 introduces and discusses the theory of Society of Mind. It lays down a broad framework for explaining intelligence and building its models. In section 3, we explain and explore Subsumption Architecture, a behavior based model that was quite successful in building simple and mostly reflexive robots. Introduction to Reinforcement learning, a concrete mathematical approach to unsupervised learning is given in section 4. Conclusions and future work that can be done to improve these theories are presented in section 5.

2 The Society of Mind

2.1 Introduction

This theory was put forward by Marvin Minsky in his famous book - *The Society of Mind* [1]. This theory puts forward ideas that were developed through many decades of careful thinking about how minds, natural and artificial, might work. It tries to explain how mind can be explained by collections of simple processes and their interconnections and dependence. Though, this theory talks about human mind most of the time but the same ideas can very well be applied in robotics to make their control systems. Careful thinking will tell us that our *mind* is nothing but a control system for a complex machine - Humans. Humans are the best examples of intelligent systems we have encountered so far. So it seems only natural that we should study human mind and find out how it works and apply those ideas to build better robots. The ideas given in this theory are essentially vague. They give a broad outline of organization of mind which enables to solve and do such vast variety of tasks and problems. This theory has incorporated and unified many ideas from psychology, artificial intelligence and computer science.

2.2 Philosophy and Assumptions

This theory assumes that brain is functionally modular and useful generalizations can be made about the operation of such modules and the inter module interactions. In order to understand mind it is essential that we describe whole working of mind in terms of simpler things that has no intelligence of its own. If any of these simpler *particles* are intelligent, we have to explain in turn their intelligence. Thus we will be caught in a circular loop. This theory concentrates on processes and tasks that are termed as *easy*, obvious or *common-sensical*. Over years many workers in the field of Artificial intelligence have found out that these tasks are the most difficult to program in a artificial creature. In general, we are least aware of things that out minds do best. Another important assumption is that people are nothing more than extremely complex machines or robots. The cornerstone of Minsky's theory is the conception of minds as collections of enormous numbers of semi-autonomous, intricately connected agents that are themselves mindless. These agents by themselves cannot perform any thought processes, but when combined into "societies", true intelligence arises.

2.3 Agents, Agencies and Societies

As mentioned earlier, for any theory that tried to explain mind, it is essential that the *particles* that those theories divide the mind into are themselves mindless and unintelligent. These particles should be smaller and simpler than anything we would consider as smart. The particles that this theory has divided mind into are known as *Agents*. Agents are the building blocks of our mind. More formally as described in [1], page 326, Agent is:

> Any part or process of the mind that by itself is simple enough to understand - even though the interactions among groups of such agents may produce phenomena that are much harder to understand.

Consider, for example, a child trying to build towers from blocks. To do this complex activity many agents inside that child would be active. A few of these agents and their functions are:

1. **BUILDER**: Build a tower.
2. **BEGIN**: Start building a tower.
3. **ADD**: Add to the height of already built tower.
4. **FIND**: Find a block to add in the tower.
5. **GET**: Retrieve a block.
6. **PUT**: Puts the block in hand on the tower.
7. **SEE**: Finds a block.
8. **GRASP**: Pick up a block.
9. **MOVE**: Moves the block held in hand.

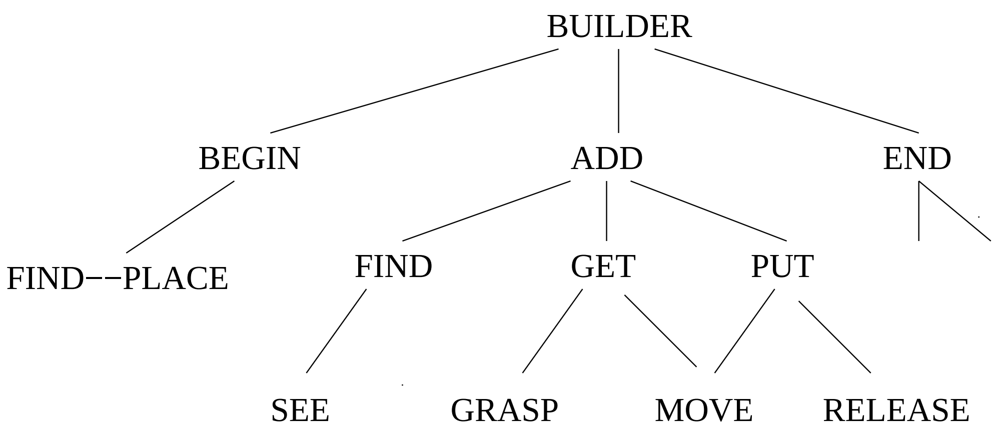

Figure 1: Agents in a Bureaucracy

10. **RELEASE**: Releases the block.
11. **END**: Finishes building tower.
12. **FIND-PLACE**: Finds appropriate place for the tower.

Each of these agents has a goal. Whenever that agent is aroused, it will try to achieve its goal. From this example of agents it is not clear that these agents are *simple*. For example, work of BUILDER agent seems quite complex. The catch is that these agents are not isolated. They are interconnected with each other and possibly other agents to do their work. They are taking help of other agents to achieve their goal.

As a agent BUILDER does no intelligent work but merely turns on BEGIN, ADD and END. In turn ADD just orders FIND, PUT and GET to do their jobs. Thus the agents are arranged into hierarchies. Following figure explains the hierarchy of above agents.

There is no complexity in agents itself. It arises from the complex networking of these agents. Each agent has a dual nature. If we see their goals or tasks, it seems that the agent *knows* how to perform those tasks. But as we know, in agents like BUILDER there is no knowledge of goal. It merely does a administrative work of activating its subordinate agents. A agent and its connected agents work together as an *agency* to perform its goals. An Agency is defined as any assembly of parts considered in terms of what it can accomplish as a unit, without regard to what each of its parts does by itself. The descriptions of goals of agents given in the example above can than be thought of as description of goals of the agencies those agents are in charge of. Thus the job of building a tower is done by whole BUILDER agency.

There are enormous number of such agents and their agencies. They constitute the Society of Mind.

2.4 Learning

A Society is not just a collection of individuals. Its their mutual interaction that fully describes a society. Similarly in order to understand about society of mind, we have to study the interaction between the agents. Until now we have just saw that agents are connected by some links. If we consider those as non-changing links, it is clear that this model can not account for learning.

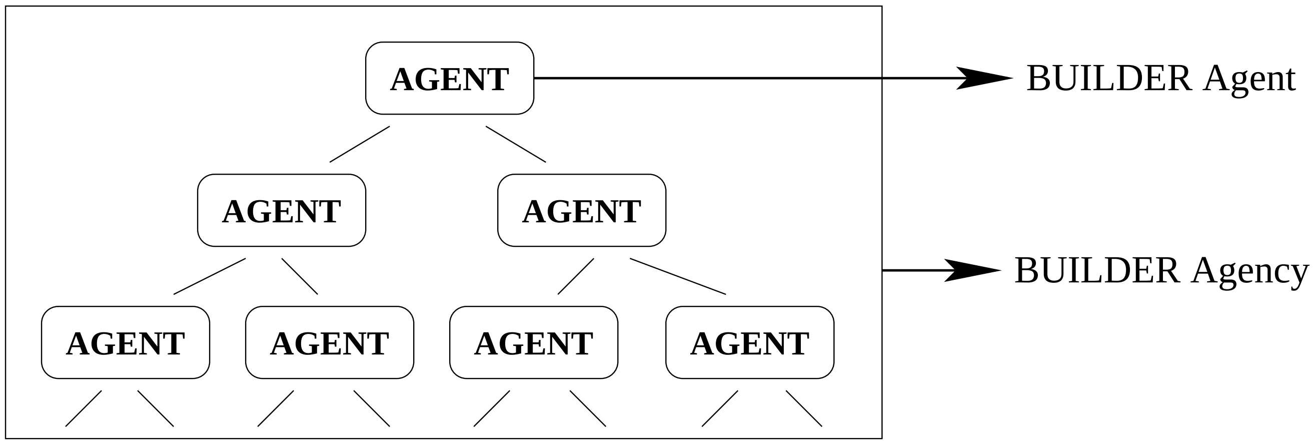

Figure 2: Dual nature of BUILDER

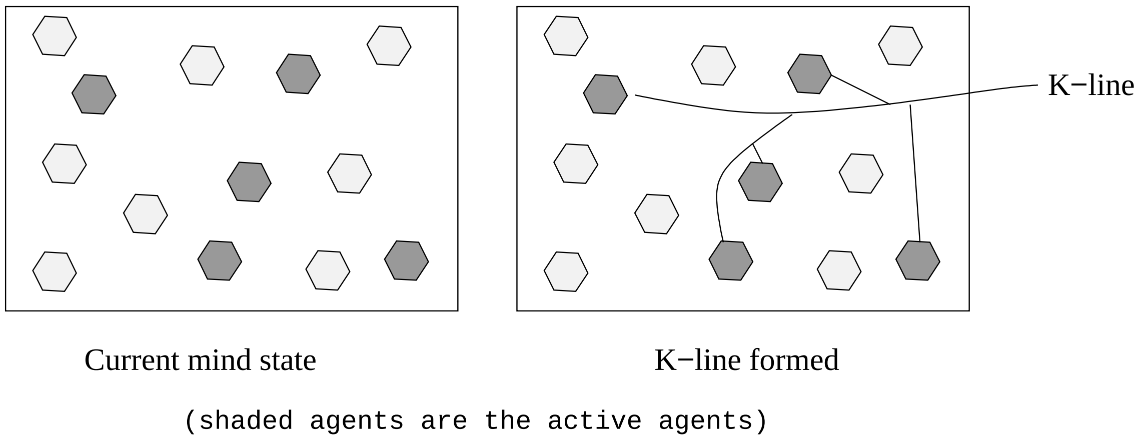

Figure 3: Formation of Knowledge-lines

Most of the knowledge we humans learn is unsupervised. But in most of problem solving there is a concept of reward. Even when we are not learning from someone we are still learning from reward. For solving problems we must make use of knowledge gathered from previous experiences. Let us say a particular agent (say A) is solving a problem. For doing so, let us assume it aroused a set of other agents. When the problem is done, the reward makes A to reinforce the connections from A to the set of agents. Next time when A is solving the same problem, these set of agents will be more likely to be aroused again by A. This reinforcement will be treated more formally when we present the Reinforcement Learning model. The reward is passed down from agent A to those sets of agents and this process continues. This way, as the intelligence system gains more and more experience the Society of Agents modifies their inter-connections and tries to evolve into a more organized and useful society.

2.5 Theory of Memory

Another important property of intelligent systems is memory. They keep track of things they have done in the past. We will now see how knowledge is represented, stored, retrieved and used in this model. This theory introduces a concept of a type of agent called a *Knowledge-line* or *K-line*.

Whenever we *get a good idea*, solve a problem, or have a memorable experience, we activate

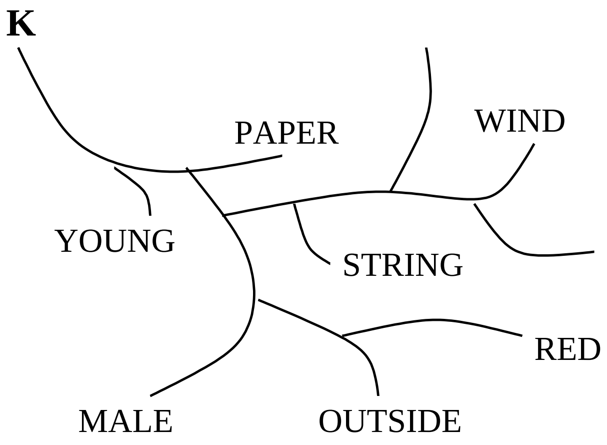

Figure 4: **Disorganized**: K-lines attached to many agents

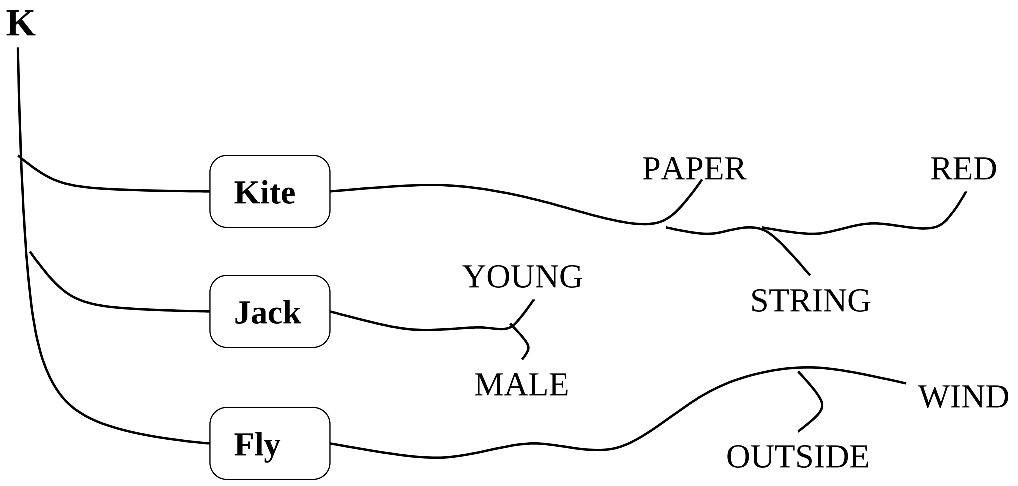

Figure 5: **Organized**: K-lines attached to K-lines

a K-line to *represent* it. A K-line is a wireless structure that attaches itself to whichever mental states are active at that time. At some later time, if we suspect that the current problem in hand is too similar to some old problem who's K-line we have already formed - we activate that K-line. Activation of a K-line automatically activates all the agents that were attached to kind. Thus we get into a mental state that is similar to the state when we solved that problem. It makes it relatively easy for the intelligent system to solve the current problem. These K-lines do not bind to all the agents with equal firmness. It makes strong connections at a certain level and weaker connections at higher and lower levels of details. This prevents overwhelming of our mind because of too many details by avoiding not so useful agents.

Suppose a K-line describing a event like **Jack flying a kite** is made. As shown in figure this K-line will attach itself to the agents active at that instant of mind state.

But when this K-line is made, suppose we already have K-lines for individual things like Jack, Kite and flying. It would be wise if instead of the original attachments this new K-line attaches itself to already made K-lines as shown in following figure. As these K-lines are also agents, we are not introducing anything new in our model.

This new scheme also has the advantage of making the memories and knowledge more hierarchical.

2.6 Criticism

As mentioned earlier, this theory assumes that

1. the brain is functionally modular
2. useful generalizations can be made about the operation of each module and interaction of sets of modules.

While there may be some evidence that assumption 1 is true, there is little evidence that the second assumptions are true. There is every possibility that even if the brain is modular, that each module behaves in an entirely idiosyncratic manner, and every pair of interacting modules interact in an entirely idiosyncratic way.

3 Subsumption Architecture

This approach for building mobile robots was put forward by Rodney A. Brooks in his famous paper [2]. The subsumption (or 'Brooksian') architecture is predicated on the synergy between sensation and actuation in lower animals such as insects. Brooks argues that instead of building complex agents in simple worlds, we should follow the evolutionary path and start building simple agents in the real, complex and unpredictable world. It was quite successful for controlling simple mobile robots.

3.1 Motivation

As mentioned earlier, this architecture was largely inspired from simple biological systems. It tried to follow the same path as nature has followed in course of evolution. Following were some important and essential goals of this theory:

- *Multiple Goals* In any real life situation, often the robot has multiple goals, some of which are conflicting. Some goals will have higher priorities based on the context. It is important that the robot's control system is responsive to high priority goals while still serving the lower level goals. The conflicts should be resolved in such a way that in whole a unified behavior of robot emerges.
- *Multiple Sensors* The robot (as most biological systems) will most likely have multiple sensors. All sensors have an error component in their readings. They will often give inconsistent readings. The control system must take *proper* decisions under these conditions.
- *Robustness* Robustness is an essential goal if are making a reliable and dependable robot. The robot must be able to adapt and carry out its goals even of some of the sensors or systems fail. When the environment changes drastically it should be able to still achieve some modicum of sensible behavior.

In addition to these goals, *additivity* would also be desirable. Once a robot is built, it is desirable that we should be able to increase its power and capabilities without much change in its internals i.e. We must be able to add new functional units without much alteration of already debugged and properly working functional units.

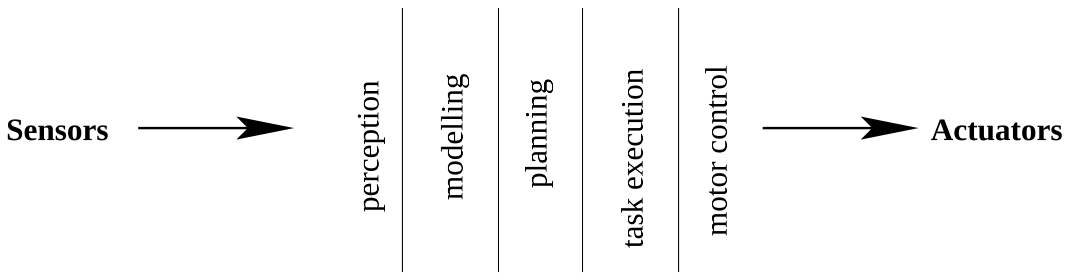

Figure 6: Traditional decomposition

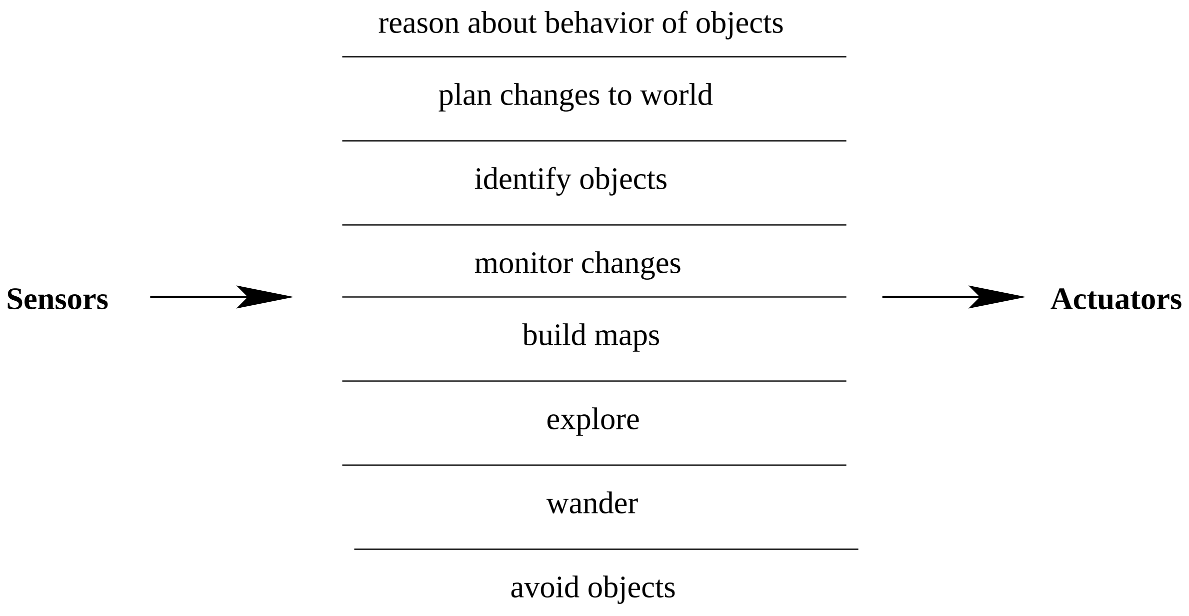

Figure 7: Brook's decomposition

3.2 Traditional decomposition into functional modules Vs. Decomposition based on task achieving behaviors

Figure 6 captures the essence of traditional decomposition of robot control systems. In this, each functional module takes its input from the right hand side module and passes its output to the left hand side module. This decomposition does not fit very well with the evidence from biology and evolution. Achieving many of the goals mentioned above is difficult in this approach as compared to approach of subsumption architecture. In the traditional approach most of the time is spend on implementing higher level behaviors, whereas evolution suggests that higher behaviors such as problem solving behavior, language, expert knowledge and application, and reason are all rather simple once the essence of being and reacting are available [3]. So Brooks suggested task achieving decomposition instead of the traditional approach as given in figure 7.

3.3 Methodological Assumptions

The philosophy and assumptions of subsumption architecture is given below. Some of these assumptions arise naturally from the desired goals mentioned in the motivation section.

- *No central model of the world* The approach of making a central representation of the world is inspired by the *Symbol System Hypothesis*, [4]. This hypothesis states that intelligence operates on system of symbols. The symbols loosely represent the actual objects in the world. The reasoning system operates in a domain independent way on the symbols. The perception system delivers the description of the outer world to the central system in symbolic form. Thus, naturally, the world model that is built must depend on kind of task that the intelligent system is carrying out. For example for a mobile robot which is just avoiding obstacles, the height of obstacles is irrelevant. But for control system of an autonomous mobile helicopter, the height of obstacles must form part of central world model. In some robots, it may even be desirable to have two sets of different representation of the same world because of variety of tasks it has to perform. Also symbol systems in their purest forms assume a non-ambiguous clear picture of the world without noise, which is definitely not the case for robots working in the real world. It is only with much computational complexity that problems like ambiguity and noise can be incorporated in this system. This leads to computationally expensive cumbersome systems.

It is because of these shortcomings Brooks in [3] rejects symbol system hypothesis as the basis of intelligence. The other approach that is suggested is the Physical grounding hypothesis. This hypothesis states that to build a system that is intelligent it is necessary to have its representations grounded in the physical world. The world is its own best model. Once this key fact is recognized there is no need for central representation of world. The modules in the system extract all the information through sensors and give all its *desires* and goals as physical actions. Thus in subsumption architecture no central world knowledge is maintained.

- *Distributed Behavior* There is no concept of central module in subsumption architecture which decides the correct behavior of the system. The whole behavior of the system emerges from unification of the task achieving behaviors of individual modules. Subsumption architecture is parallel in the sense all the modules behave independently of each other. This assumption goes in a long way in satisfying the goals of additivity and robustness.

- *Organization of Agents* The Agents are organized in a bottom-up fashion. Each agent is not aware of agents in the layers above it. Agent at a particular level is not allowed to directly interfere with the working of agents at lower level. It can modify their behavior only by altering their inputs and outputs. Thus complex behaviors are fashioned from the combination of simpler, underlying ones.

3.4 Architecture Description

As mentioned in the previous section, the agents or the tasking achieving behaviors are organized in a bottom up layered fashion. As explained in figure 7 the problem is decomposed vertically into layers of desirable external behaviors of the robot control system.

3.4.1 Levels of Competence

A *Level of Competence* is an informal specification of a desired class of behaviors of a robot over all environments it will encounter. A higher level of competence implies a more specific desired class of behaviors. For example, the levels of competence for a mobile robot could be [2]:

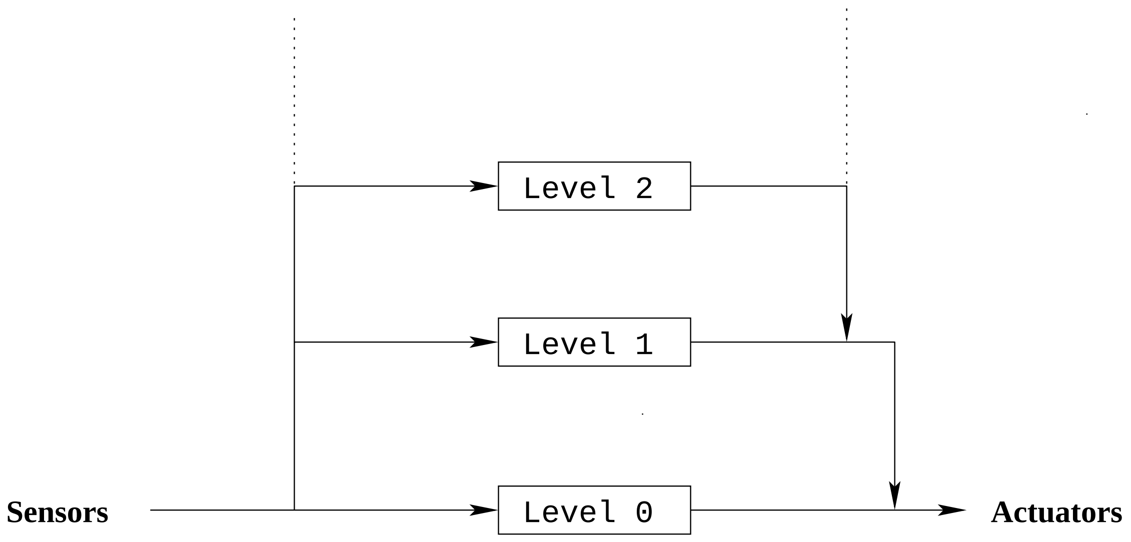

Figure 8: Layered control system

1. Avoid contact with objects (whether objects move or are stationary).
2. Wander aimlessly around without hitting things.
3. *Explore* the world by seeing places in the distance which look reachable and heading for them.
4. Build map of the environment and plan routes from one place to another.
5. Notice changes in the *static* environment.
6. Reason about the world in terms of identifiable objects and perform tasks related to certain objects.
7. Formulate and execute plans which involve changing the state of world in some desirable way.
8. Reason about the behavior of objects in the world and modify plans accordingly.

Here each level of competence includes the earlier levels of competence as its subset. Each level of competence further constrains the set of valid behaviors. This is very much motivated by Biological evolution. The robot with a implementation up-to a particular level of competence is identical to a organism at a step in the big ladder of evolution.

3.4.2 Layers of Controls

For each level of competence we build a layer of control. Higher level of competence is achieved by adding a new layer to the already added layers. As shown in figure 8 each layer of control is allowed to examine the data from level 0 system and also allowed to inject data into the internal interfaces of level 0 suppressing the normal data flow. A lower layer continues to run unaware of the higher layer which may sometimes interfere in its data paths. A major advantage of this is that the system can be partitioned at any level, and the layers below form a complete operational system. Thus this provides a way to incrementally build and test a complex mobile

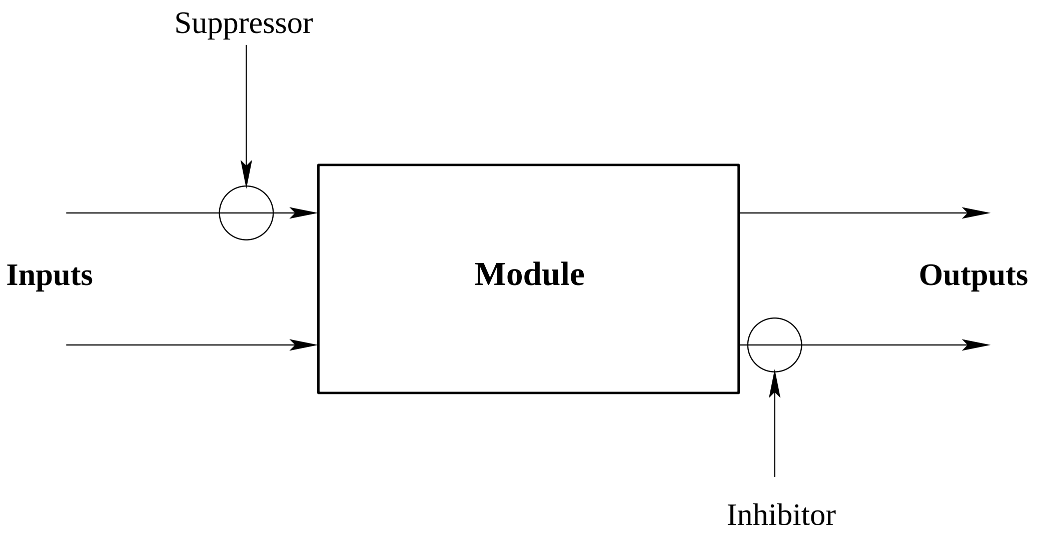

Figure 9: Suppression and Inhibition

robot control system.

Each layer is made of set of small processors which send messages to each other over connecting wires. Each processor is a augmented finite state machine (AFSM) with a set of registers for holding data and retriggerable monostables. These processors run completely asynchronously, monitoring their input wires and sending messages to output wires. A new message overwrites any existing message on that *wire*. Thus it is possible for messages to get lost. The arrival of a message, or the expiration of a timer, can trigger a change of state in the finite state machines. Finite state machines states can either wait on some event, conditionally dispatch to one of two other states based on some combinational predicate on registers, or compute a combinational function of the registers directing the result either back to one of the registers or to an output of the augmented finite state machine. All clocks of the subsumption system have same tick period which is known as the *Characteristic frequency* of the particular subsumption implementation. These clocks need not be synchronized.

3.4.3 Communication between layers

The AFSM's have some input lines and some output lines. An output line from a module is connected to input lines of one or more modules. These are the only ways in which AFSM's can communicate with each other. In addition to these, there are three other concepts of communication - Inhibition, suppression and defaulting.

As shown in figure 9 a output can be inhibited. An extra wire can tap into the output of a AFSM. If a signal arrives from this wire it inhibits the output of the AFSM for some small time period. This time period is usually equal to two clock ticks of the internal clocks. The original message sent to this wire during this time period is lost.

Similarly, we can tap into the input of a AFSM. This is called suppression. Its behavior is similar to inhibition. The signal from the tapping wire is fed into the module for a small time period. Defaulting is like suppression, except that the original wire, rather than the new side-tapping wire, is able to take control of the messages sent to the destination [13]. In all these cases, a continuous supply of suppressing messages is required to maintain control of the side-tapped wires. It is by these mechanisms that a conflict between different behaviors is resolved.

[5] describes the *Behavioral Language*, a rule-based real-time parallel robot programming language which compiles into the subsumption architecture just described.

3.5 Extensions to Subsumption Architecture

The subsumption architecture model described here is essentially reflexive. It gives a rapid and reflexive response to its environment. But due to only reflexive response, subsumption architecture is not totally *salient* and *adequate*. To overcome these limitations, some extensions to the original subsumption architecture has been proposed.

3.5.1 Hormonal Activation

As we have seen, Subsumption Architectural agents may be considered partially salient. There are conflicts between different kinds of behaviors. At a given time, many behaviors would be competing for actuator resources. It is essential that conflict should be resolved in favor of behavior which is appropriate at that moment.

To resolve this problem, Brooks in [6] proposed hormonal activation as an extension to the subsumption architecture. This model is inspired by model of biological hormone system given by [7]. Hormones in biological system are kind of low bandwidth global communication scheme. But they are not controlled by any centralized agent. Their release and activation of behaviors by them happens purely locally. This fits with the overall idea of subsumption architecture.

In this model any sensory input do not directly release hormones. Instead a two stage mechanism is used for hormonal activation of behaviors.

Conditions Any computational process can excite a condition. Examples of conditions is *panic* or *drowsiness*. Many processes can excite same condition. In this case their excitation is added. Conditions have a value between 0 to 15. Any process can excite a condition by giving any increment to its current value. The conditions decays according to the function (linear, bilinear) provided by the programmer. These conditions do not directly affect any behaviors. This is done by another set of *Releasers*.

Releasers Releasers are the one which directly effect the behaviors. They are more closely related to hormones. A releaser's value at any instant is a function of activation values of a set of conditions.

Affect of Releasers on behavior A behavior is a collection of Augmented Finite State machine. A AFSM is a computational process. Each behavior has associated with it a *Activation level*. A behavior becomes active when the activation level crosses some threshold. The activation level of the behavior is further function of one or more releasers. As mentioned above a behavior can have two states - active or inactive. Based on the state of behavior the processes within them are allowed to run. Processes are further divided into three classes:

1. *Regular Processes:* These processes always run, and any message they send out reach their destinations.

2. *Haltable processes:* These processes only run when the behavior is active. Otherwise they do not run, do not process their inputs and do not send out any messages.

3. *Inhibitable processes:* These processes always run but when the behavior is inactive their outputs are inhibited or blocked. They still receive all their inputs however and can retain and change internal state.

In this way, a global integration of behaviors is done without any use of a global centralized agent. Activation priorities of two conflicting behaviors can be decided by such kind of hormonal states. In many cases, the higher level behaviors may want to turn off the lower level survival behaviors because they have access to more sophisticated information. Without any need for explicit knowledge, the system will have knowledge of global state. It will *know* what it is trying to do and a global unified and consistent behavior will emerge.

3.5.2 Planning and Learning

In the original subsumption architecture there is no provision for planning and learning. The robot built using this architecture would be totally reflexive. In many real life situations, it is desirable and often essential that the robot learns from the world and when trying to achieve its goal(s), it should plan properly to maximize output and minimize usage of essential resources. The learning part can be achieved by keeping some selective higher level agents which are not reflexive and which maintain a global learning database. But this goes against the whole idea of not having any centralized agent and any centralized world model.

To resolve this problem, [8] proposed a model of planning and learning which is totally reactive and built bottom up. This model is totally compatible with the original philosophy of the Subsumption architecture. In this model, the representation of the entire system is homogeneous. It consists of simple reactive rules which encode both the control strategy and the knowledge of the system. The processes itself encapsulates the knowledge of the robotic system.

This model is explained further with help of example of a mobile robot which builds map of landmarks it encounters (learning) and then plans its way to a specified goal (planning).

As figure 10 illustrates, in the lower layer has the basic behavior of the system - navigation. The boundary following behavior of the system adds navigational reflexes to the system which keeps the agent close to potential landmarks. landmark detector then utilizes the boundary following behavior of the navigational system to detect potential landmarks in the system. The landmark detector consists of set of monitors which keep track of relevant features of the environment and increase or decrease landmark detected confidence level. When this confidence level increase beyond a specified threshold a landmark is detected and sent to the map module.

In the map module, Learning constructs and constantly updates the topological map of the environment. The map is a topological network of processes, each of which corresponds to a landmark in the world. Whenever a landmark is detected by the landmark detection module, it is broadcast to all processes in the map. If none of the processes recognizes it, the landmark is added to the map as new. The topology of the map is maintained isomorphic to the environment. A landmark is described by its type, provided by the landmark detector, compass heading and its topological position. On finding a new landmark, a process is associated to it and it is linked to its neighboring physical landmarks via communication links. If a landmark is recognized by

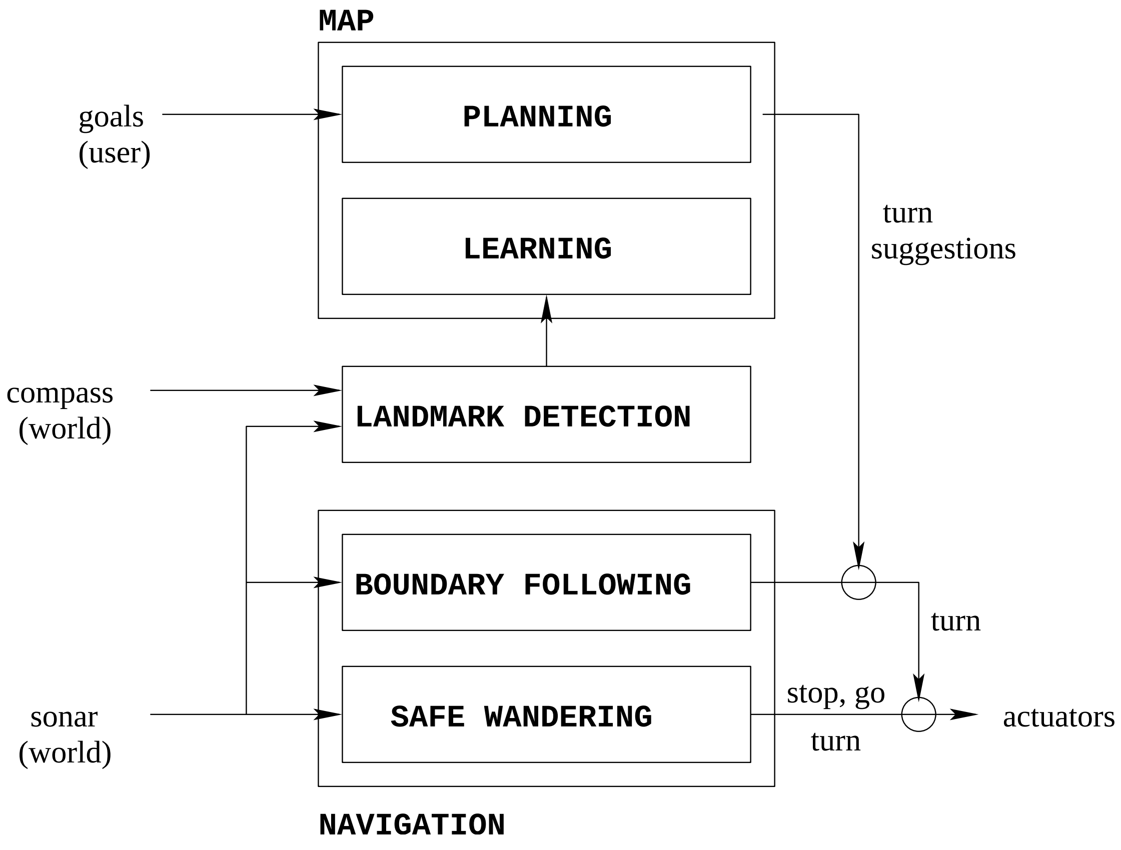

Figure 10: Map building and planning agent: capabilities and information flow

a process, it gives us the position of robot in the map.

Planning is also implicitly done by the system through the map module. Whenever the user selects a goal location, the process associated with the landmark of the goal becomes activated. Each process in turn spreads its activation to its neighbors. The activation decays down as function of physical distance. Thus to plan the shortest path to the goal, the agent has to just see the activation values of the nearest landmarks. This turn suggestions are then given to the actuators by the map module by inhibiting the output of the navigation module.

Thus the whole topological map of the environment is implicitly built into the system without any need of central world model. This model was again inspired by biological systems. This model is a plausible interpretation of known physiological data of rat navigation [9].

3.6 Criticism of Subsumption architecture

1. Subsumption architecture is not sufficiently modular. Upper layers interface with internal functions of lower layers. To design the upper layers we cannot treat the lower layers as black-boxes. As the complexity and size of the system will increase it will become more and more difficult to *code* the higher layers of controls. The entire framework requires building feedback loops by hand, with no theory of how to construct them.

2. Subsumption architecture was inspired by lower biological systems such as insects. The abilities and intelligence of humans is definately much more superior than insects. But it is highly unlikely that the same architecture can explain human intelligence also.

3. It is not clear that how human abilities like Language can be explained without maintaining

any central model of world.

4 Reinforcement learning

4.1 Introduction and Motivation

When a child comes to the world, his intelligence agency does not know which behaviors to perform. Initially, his actions are random and which each action he often gets some reward or punishment (negative reward). As he gains more and more knowledge about which actions gives rewards and which actions give punishment, he modifies his behavior to get more reward. This process, which starts at childhood, continues throughout the life of an individual.

In many real life situations, this is exactly the position a robot finds itself in. The programmer cannot imagine all the situations the robot will face, and decide the optimal action for the robot. In applications like unmanned planetary exploration, it is quite likely that robot finds itself in a situation in which it does not know the optimal action to take. The robot will be aware of its goal but not about the action that will take it to its goal. We would ideally like that the robot, like humans, explores different actions and based on its experience, tries to find the optimal action.

One such approach of learning by interaction with the environment is *reinforcement learning*. In reinforcement learning problem, there is a learner that is connected to its environment through sensors and actuators. The learner knows what to do - It has to maximize the rewards given to it by the environment. But it does not know which actions will maximize the reward. It may start by trying a random action and noting the reward that it gave. After many such trials, he will know which actions give more rewards. It has to direct its future actions based on this information so as to maximize the total reward it receives from the environment in the long run.

From above discussion it is clear that the agent must have two properties of exploration and exploitation for learning. At a given time, to obtain more reward, the agent must prefer actions that in the past have given him more rewards. The agent has to exploit the information it has already have to increase the reward, but to get this information it is essential that it explores the actions it has not selected before and finds out the rewards they give. It is clear the if the agents behavior is biased too much toward one of these properties, there will not be any learning. So a appropriate trade-off between these two behaviors must be decided.

4.2 A formal specification of the Reinforcement Learning problem

4.2.1 Agent and its Environment

The whole concept of reinforcement learning is based on interaction between a learner and its surroundings. The learner is known as the *Agent*. Everything surrounding it which is not in its direct control is called its *Environment*. The Agent can only sense the Environment through its sensors. It cannot arbitrarily change the environment. It can only specify some actions to be taken in response to a particular state of environment. For example, in mobile robots action could be the direction and velocity in which to move. The environment also gives rise to reward and the agent's goal is to maximize it over long run.

The figure 11 shows this relationship between the agent and its environment. The Agent and the Environment interact at time steps, $t = 0, 1, 2, 3....$ At each time step t, the agent gets a representation of the environment's state $s_t \in S$, where S is the set of possible states. In the

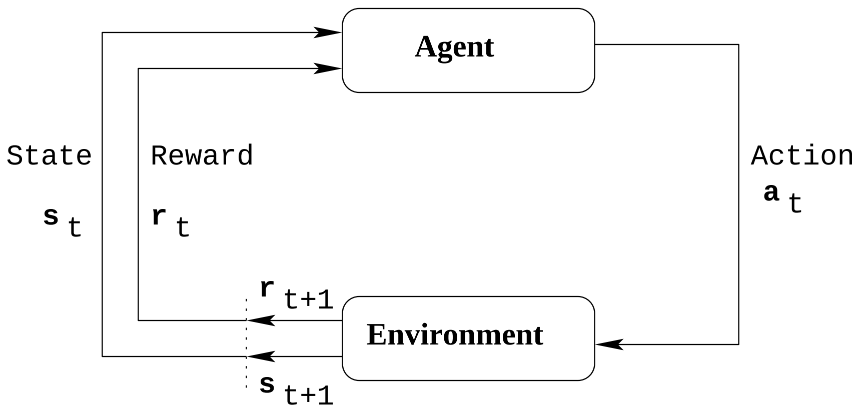

Figure 11: Agent-Environment interaction

robotics example, s_t can be the collection of sensors readings or some processed value of those readings which provides us with the complete description of the current state. The Agent, in response to state s_t, selects a *action*
$a_t \in A(s_t)$, $A(s_t)$ is set of available actions in the state s_t.
This action of the agent will change the environment's state from s_t to s_{t+1} one time step later. Also, in addition, the agent receives a numerical *reward* from the environment, $r_{t+1} \in \Re$.

In many cases, rewards are also computed inside agents, but as shown in the figure they are considered external to the agent as the part of environment. This is because the rewards are not in direct control of the agent and we consider anything that is not in direct absolute control of the agent as its environment. Also the representation of the environment's state s_t and agents action a_t will vary from problem to problem.

4.2.2 Relationship between Goals and Rewards

As we have mentioned earlier, in reinforcement learning the goal of the agent is maximize the total reward it gets. The agent should make its decisions such that the reward in long run is maximized not just the immediate reward.

The agent is not directly aware of the goal the programmer wants it to achieve. It's goal is only related to maximizing its reward. Thus it is the duty of programmer to choose the reward carefully and cleverly so that in maximizing its reward the agent achieves its goal. For example, in case of a robot learning to move without bumping into nearby objects, the reward can be 1 if it does not bump into anything. Whereas the reward should be -1 in case it bumps into any object. For a robot trying to escape from a given maze as soon as possible, the reward can be -1 until it escapes from the maze and on escaping it can be given a reward of $+1$.

In all these examples, the agent will eventually learn to maximize the reward. Doing this will require agent to formulate its behavior which allows it achieve the goal the programmer wishes it to achieve. The reward is the programmer's way of communicating to the agent what he wants to achieve. The reward should never be used to give information such as how the programmer wishes to achieve its reward. This is a common mistake. For example, many times we know some good heuristics to solve a problem. If reward is given so that agent proceeds according to these heuristics, it is quite possible the agent may learn to maximize its reward without even actually achieving the final goal. So its important that programmer only gives information about *what* is to be done, not *how* it is to be done. If the reward is properly defined for the goal, in the long run, the agent will actually discover the heuristics.

4.2.3 Returns

As we have mentioned the agents goal is to maximize the reward in long run. But we have not specified its precise meaning. If the agent is at time step t, there is a sequence of rewards r_{t+1}, r_{t+2}, r_{t+3}, $\ldots$ after this state. We have not specified what aspect of this sequence the agent wants to maximize.

We want the agent to take into account the future also while making current decisions. This is done through the concept of *expected return.* The return, R_t, is defined as some specific function of the reward sequence. In the simplest case, we can define R_t as

$$R_t = r_{t+1} + r_{t+2} + r_{t+3} + \ldots + r_T$$

Here T is the final step. The concept of final state makes sense in situations which the whole agent-environment interaction can be broken into *episodes.* After each episode, there is a reset to a initial state. An examples of such interaction is agent playing game such as chess or a robot trying to escape from a maze. These kind of tasks are called *episodic tasks.*

It is not always possible to break the interaction into identifiable episodes. This is the case for a robot, which is continuously exploring planetary surface or enemies territory. There is no end state in these kinds of task. These kinds of tasks are called *continual tasks.* The notation for episodic and continual tasks can be unified by considering a special terminal state in the episodic tasks which loops to itself and from that state on-wards the rewards are 0. Thus it would not make a difference if we sum over first T steps or over the full infinite sequence.

The model of return given above is obviously not suitable for continual tasks because as $T = \infty$, the return could become infinity. For this additional concept of *discounting* is used. In this approach the more distant is a reward from current state, the more heavily is it discounted. The discounted return function is defined as

$$R_t = \sum_{k=0}^{\infty} \gamma^k r_{t+k+1}$$

where γ is called the discount rate, $0 < \gamma < 1$.

Thus a reward received after k steps seems only $\gamma^{k=1}$ times its worth if it was received immediately. γ give us the degree of far-sightedness of the agent. As γ increases the agent becomes more and more far-sighted.

Another approach is the *average-reward model*, in which the agent is supposed to optimize the reward function

$$R_t = \lim_{h \to \infty} \frac{1}{h} \sum_{k=0}^{h} r_{t+k+1}$$

In this model, it is impossible to distinguish between the cases when the reward comes in early phases and the case when it comes later. It is possible to generalize this model so that it takes into account both the long run average and the amount of initial reward than can be gained. In the generalized, *bias optimal* model, a policy is preferred if it maximizes the long-run average and ties are broken by the initial extra reward.

4.2.4 Markov Decision Processes

An environment satisfies the Markov property if its state compactly summarizes the past without degrading the ability to predict the future. This is rarely exactly true, but often nearly so; the state signal should be chosen or constructed so that the Markov property approximately holds. If the Markov property does hold, then the environment is called a *Markov decision process* (MDP). A *finite MDP* is an MDP with finite state and action sets.

We can define a particular MDP completely by its state and action sets and by the one-step dynamics of the environment. Given a state s and an action a, the probability of next state s' is

$$P^a_{ss'} = Pr(s_{t+1} = s' | s_t = s, a_t = a)$$

Also, the expected value of reward, given a state s, action a and next state s' the expected value of reward is

$$R^a_{ss'} = E(r_{t+1} | s_t = s, a_t = a, s_{t+1} = s')$$

We can see that these two quantities completely specificy all the important acpects of the dynamics of a finite MDP we are interested in.

4.2.5 Value Functions

We define *policy* π as a mapping from states, $s \in S$, and actions, $a \in A(s)$, to the probability of taking action a when in state s. Value function $V^\pi(s)$ (the value of a state s under a policy π) is the expected return when starting in state s and following the policy π thereafter. For MDPs and discounted return function, $V^\pi(s)$ can be defined more formally as:

$$\begin{aligned} V^\pi(s) &= E_\pi\{R_t | s_t = s\} \\ &= E_\pi\{\sum_{k=0}^{\infty} \gamma^k rt + k + 1 | s_t = s\} \end{aligned}$$

where E_π denotes the expected value given that the agent follows policy π and t is any time step. The function V_π is known as the *state-value function for policy* π.

Similarly, the *action-value function for policy* π, Q_π, which is the expected return starting from s, taking the action a, and thereafter following policy π, is defined as:

$$\begin{aligned} Q^\pi(s,a) &= E_\pi\{R_t | s_t = s, a_t = a\} \\ &= E_\pi\{\sum_{k=0}^{\infty} \gamma^k r_{t+k+1} | s_t = s, a_t = a\} \end{aligned}$$

The value functions defined above can be easily estimated through experience. If the agent keeps track of the actual returns following a state for each state, and maintains a average then it will be a good approximation of , $V_\pi(s)$. It can be seen that as the number of times a state is reached approaches infinity this average converges to the state value $V_\pi(s)$. Similarly if we maintain separate averages for each action taken in a state, we can get estimation of $Q_\pi(s)$.

For any policy π and any state s, the following consistency condition holds between the value of s and the value of its possible successor states:

$$V^\pi(s) = E_\pi\{R_t | s_t = s\}$$

$$\begin{aligned}
&= E_\pi\{\sum_{k=0}^{\infty}\gamma^k r_{t+k+1}|s_t = s\} \\
&= E_\pi\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s\} \\
&= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma E_\pi\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s] \\
&= \sum_a \pi(s,a)\sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^\pi(s')]
\end{aligned}$$

where $a \in A(s)$ and $s \in S$.
This equation is called **Bellman equation** for V^π. The value function V^π is the unique solution to its Bellman equation. This equation forms basis of number of ways to compute, approximate, and learn V^π.

4.2.6 Optimal Value Functions

Solving a Reinforcement learning problem is equivalent to finding a policy that achieves a lot of reward over long run. But given a correct model, how to find that optimal policy? A policy π_1 is called better than another policy π_2 if and only if

$$V^{\pi_1}(s) \geq V^{\pi_2}(s)\forall s \in S$$

There is always at least one policy that is better than or equal to all other policies. Such a policy is called optimal policy and set of all optimal policies is denoted by π^*. The optimum value function and optimum action-value function for π^* is defined as:

$$V^*(s) = \max_\pi V^\pi(s)\forall s \in S$$
$$Q^*(s,a) = \max_\pi Q^\pi(s,a)\forall s \in S, a \in A(s)$$

We can now derive the *Bellman optimality equations* for V^* and Q^* whose solutions will enable us to find V^* and Q^* for a finite MDP, given its model.

$$\begin{aligned}
V^*(s) &= \max_{a\in A(s)} Q_{\pi^*}(s,a) \\
&= \max_{a\in A(s)} E_{\pi^*}\{R_t|s_t = s, a_t = a\} \\
&= \max_{a\in A(s)} E_{\pi^*}\{r_{t+1} + \gamma\sum_{k=0}^{\infty}\gamma^k r_{t+k+2}|s_t = s, a_t = a\} \\
&= \max_{a\in A(s)} E\{r_{t+1} + \gamma V^*(s_{t+1})|s_t = s, a_t = a\} \\
&= \max_{a\in A(s)} \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma V^*(s')]
\end{aligned}$$

$$Q^*(s,a) = \sum_{s'} P^a_{ss'}[R^a_{ss'} + \gamma \max_{a'} Q^*(s',a')]$$

4.2.7 Finding Policy

For finite MDPs, the Bellman optimality equation has a unique solution independent of the policy. The Bellman optimality equation is actually a system of equations, one for each state, so if there are N states, then there are N equations in N unknowns. If the dynamics of the environment are known, then in principle one can solve this system of equations for V^* using any one of a variety of methods for solving systems of nonlinear equations. One can solve a related set of equations for Q^*.

Given V^* it is easy to find the optimal policy π^*. For each state s, we will have some actions for which the maximum is obtained in Bellman optimality equation. Any policy assigning non-zero probability to these actions is an optimal policy. If we have Q^*, then for a given state the agent can directly choose the action that maximizes $Q^*(s,a)$.

Thus reinforcement learning problem can be solved by solving bellman optimality equations. This approach is rarely used in practice. The major reason of this is the huge need for computational and memory requirements. Because of this problem, we have to rely on approximate solutions. There are many methods in reinforcement learning that can be clearly understood as approximately solving bellman's equation. Examples of such methods are Dynamic programming, monte carlo and temporal difference learning methods [10].

5 Conclusion and future work

Though Subsumption architecture has worked well for *simple* reflexive robots, it is not clear that whether the same architecture, without any modifications, can be applied for solving more complex and big problems. Systems programmed by Subsumption architecture are purely reflexive, which makes them fast. Extensions to this architecture have added some representation without losing the robustness and reactivity of pure subsumptive architectures.

Currently the behavior generating modules and the interaction between these modules is hand-coded and hard-wired. If this process is automated or semi-automated, we can easily build larger, complex and more competent systems.

There are a variety of reinforcement-learning techniques that work effectively on a variety of small problems. But very few of these techniques scale well to larger problems. It is very difficult to solve arbitrary problems in the general case.

One of the problems with reinforcement-learning is that the agents have a hard time even finding the interesting parts of the space. They wander around at random never getting near the goal, or they are always *killed* immediately. One way to overcome this problem is by programming a set of *reflexes* that cause the agent to act initially in some way that is reasonable. These reflexes can eventually be overridden by more detailed and accurate learned knowledge, but they at least keep the agent alive and pointed in the right direction while it is trying to learn.

From the discussion above it appears that the two theories are in a way, supplementary. it would be interesting to see if the drawbacks for both these theories can be removed by somehow combining them. Some work has been already started in this direction [11].

Acknowledgements

I would like to express my heartfelt gratitude towards Dr. Pushpak Bhattacharyya for his constant guidance and encouragement without which this seminar report would have been an exercise in futility.

References

[1] Marvin Minsky, *T*he Society of Mind. Simon and Schuster, New York, 1985.

[2] Rodney A. Brooks, *A* Robust Layered Control System for a Mobile Robot. IEEE Journal of Robotics and Automation, RA-2, April 1986, 14-23.

[3] Rodney A. Brooks, *E*lephants Don't Play Chess. in P. Maes, ed., Designing Autonomous Agents, MIT Press, 1990, 3-15.

[4] Herbert A. Simon, *T*he Sciences of the Artificial. MIT Press, 1969.

[5] Rodney A. Brooks, *T*he Behavior Language: User's Guide. MIT AI Memo 1227, April 1990.

[6] Rodney A. Brooks, *I*ntegrated systems based on behaviors. SIGART Bulletin, Vol. 2, 1991, 46-50.

[7] Edvard A. Krawitz, *H*ormonal Control of Behavior: Amines and the Biasing of Behavioral Output in Lobsters. Science 241, September 30, 1988, 1775-1781.

[8] Maja J Mataric, *B*ehavioral synergy without explicit integration. SIGART Bulletin 2, 2(4), 1991, 130-133.

[9] Maja J Mataric, *N*avigating with a Rat Brain: A Neurobiologically-Inspired Model for Robot Spatial Representation. From Animals to Animals: First Intenational Conference on Simulation of Adaptive Behavior Proceedings, J. Meyer and S. Wilson, eds., MIT Press, 1990, 169-175.

[10] Richard S. Sutton and Andrew G. Barto, *R*einforcement Learning. An Introduction. Cambridge, MA: MIT Press, 1998.

[11] Sridhar Mahadevan and Jonathan Connell, *A*utomatic Programming of Behavior-based Robots using Reinforcement Learning. Artificial Intelligence , vol. 55, Nos. 2-3, June 1992, 311-365.

[12] Rodney A. Brooks, *C*hallenges for Complete Creature Architectures, in: Meyer, J.-A./Wilson, R. (Eds), Simulation of Adaptive Behavior. MIT Press, Cambridge MA, 1991, 434-443

[13] Rodney A. Brooks, *A* Robot that Walks: Emergent Behavior from a Carefully Evolved Network. Neural Computation, 1:2, Summer 1989, 253-262.