

# Using Contextual Sequences for the Control of Autonomous Robots

Victor E. Hansen

MKS Inc.

Wayne, PA 19087

## 1. ABSTRACT

This paper introduces the concept of contextual sequencing as a control structure for autonomous agents (robots). The use of context in the construction and access of a knowledge base is described. The design of a robot using a contextual evaluator is outlined. The benefits, including the ability to readily adapt, to learn, and to pretend are discussed. V4, an implementation of a contextual evaluator, is briefly described. An autonomous creature, implemented as a set of V4 rules, is given and explained as an example of these ideas.

**Keywords:** *context, contextual sequence, autonomous control, robotics*

## 2. INTRODUCTION

The word “context” is derived from the Latin *con* + *texere* meaning to weave together. The Oxford English Dictionary gives as one of its meanings “Ambient conditions, a set of circumstances; relation to circumstances”.

We, as humans, interpret virtually everything within a given context. The meanings of words, the interpretation of other people’s actions, even our own actions are all based, to a greater or lesser extent, on the weaving together of what we experience through our senses and emotions.

For instance, most people could tell you that Pegasus was a flying horse and maybe even tell you about some of its adventures. These same people would also tell you that there are no flying horses and that the very notion is absurd. These conflicting statements arise from different contexts. In a mythological context, Pegasus was very real. In the “real world” we do not have flying horses. Consider the following: “7 • 8 • 9”. Is this a sequence of numbers? Given the question, “Why is 6 afraid of 7?” (because “7 • 8 • 9”) then the sequence of numbers becomes the punchline of a joke and the digits take on completely different meanings.

The importance of context in information systems and processing is now just beginning to be realized. Many important topics in computer science can benefit from the theoretical and practical uses of context. For instance, the problem of code reusability can be reframed as the problem of using the same code in different contexts. Many modeling, forecasting, and data analysis problems can be greatly simplified with a contextual perspective.

This paper will address the use of context in the design of autonomous agents (robots). What will be shown is the benefits of a contextual model in the control systems of a robot. The major sections of the paper are-

- The characteristics of a contextual evaluator, i.e. an evaluator that interprets statements within a given context.
- How sequences of items (or actions) can be modeled in a contextual framework.
- A framework for the design of a contextual robot.
- How learning, planning, and pretending can be achieved by a robot.
- A brief overview of V4 which is an implementation of a contextual evaluator.
- A simple example autonomous “creature” implemented as a set of V4 rules.

## 3. A CONTEXTUAL EVALUATOR

A contextual evaluator can be modeled as a function with three arguments,  $f(q,k,c)$ , where  $q$  is the query to be evaluated,  $k$  is the knowledge base, and  $c$  is the context. The result is the *best answer* in the knowledge base for the given context. A complete description of this function is beyond the scope of this paper but is available from the author. Issues involving the definition of *best answer* and handling ambiguity will not be discussed here.

A contextual evaluator does have certain characteristics that make it unique. Some of these are described below-

### 3.1.1. Ability to store contradictory information

We should be able to use different contexts to store contradictory information. For instance the two statements- “The average speed over the terrain is 1.5 kt in scenario A”. And, “The average speed over the terrain is 2.5 kt in scenario B”. The query “What is the average speed?” would evaluate to 1.5 in the context of scenario A and 2.5 in scenario B.

### 3.1.2. Time

Time should be handled by the context allowing information to be stored and retrieved with a time component. Unlike the examples above, time is not a discrete contextual element but a continuous one. If we have the two facts, “The arm is at position  $A$  at time  $t$ ” and “The arm is at position  $B$  at time  $v$ ” where  $u > v$  then the query “What is the position of the arm at time  $u$ ?” where  $t > u > v$  should return position  $d$ .

### 3.1.3. Query in multiple contexts and compare results

The evaluator should be able to compare results from different contexts. In the above example, the query “What is the difference in average speed between scenario A and B?” should return “1 kt”. A more sophisticated example would be the rule “In scenario C, the average speed over the terrain

is 1 kt greater than normal.” Here we have defined a context that refers back to a prior context on which it is based.

### 3.1.4. Integration of facts, rules, & procedures

The knowledge base must be able to store knowledge as facts, rules, and procedures. For example, “The phone number of John is 555-1234” and “The phone number of anyone is obtained via procedure PhoneLookup(person).” If the phone number of John is requested then the number is immediately returned. If the phone number of anyone else is requested then the procedure PhoneLookup is invoked.

### 3.1.5. Using context to qualify a query

The evaluator should be able to take a general query and use the context to return a specific answer. For example, suppose the knowledge base contained the following- “The battery life at temperatures  $x$  through  $y$  is 900 minutes” and “The battery life at temperatures  $y$  through  $z$  is 950 minutes”. If the query “What is the battery life?” is made then the evaluator would return 900 if the context contained a temperature of  $x$ - $y$ , 950 with a temperature of  $y$ - $z$ . If the context did not specify a temperature or it was outside of the range of  $x$ - $z$  then the evaluator would fail for the query.

Another example would be in the optimization of a function of the form  $y = f(x_1, x_2, \dots, x_n)$  where the parameters of optimization are not the  $x_i$  but the context elements which affect their values. The optimization would be performed by enumerating through combinations of context elements, evaluating the  $x_i$  parameters of the function and then calculating  $y$  for each enumeration.

## 3.2 Contextual Sequences

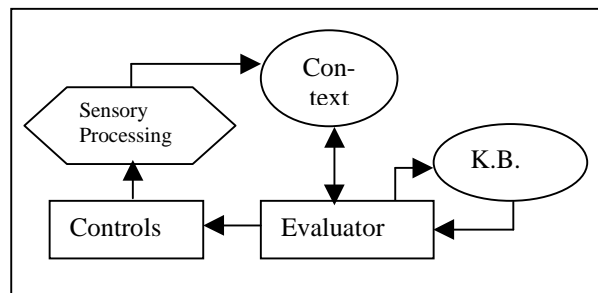
A contextual sequence is a sequence where each subsequent element of the sequence is determined by the current context. Sequences fall between two extremes- predetermined and undetermined. Predetermined sequences have a set sequence of elements and the “next” element is based on the current element (plus context). A simple example is the letters of the alphabet (“D” is the next in sequence given the current letter “C”). A more complicated sequences is a computer program where the next element (pc address) is based on the current location and the current context (condition codes, interrupt codes, etc.)

Undetermined sequences are those where nothing is pre-planned. The “next” element of a sequence is effectively unknown until it is evaluated within the current context. An example of this would be in scanning a database for records of a particular type. Here the sequence would be records or rows from the database. We do not know, ahead of time, what is coming or when the end of the sequence will occur.

We are most interested in the middle ground. These are sequences where there is an overall plan. But, conditions (contexts) can arise to cause us to deviate temporarily or permanently from the planned sequence.

## 4. A CONTEXTUAL ROBOT

A contextual autonomous robot is a robot that bases each next action solely on the current context. A block diagram would look something like-



In operation, the sensory processing unit would gather data from the outside world and the robot itself; perform some level of processing; and present the results to the context. The evaluator would continuously evaluate the *next step* given the current context and the knowledge base. Results of the step could effect controls of the robot, add new rules to the knowledge base, or update the context.

It should be noted that this form of processing could be used to implement a Turing machine. In theory, any algorithm can be implemented.

### 4.1 Learning

Learning for our robot would be done by creating new rules and injecting them into the knowledge base. These rules would be created by existing rules and new information being “received” via the context. New knowledge might be facts (“The depth of the water at coordinates  $(x,y)$  is 3 meters”) learned by the robot, or new procedures downloaded from a base station.

Learning may also be directly related to the robot itself. Assume that the robot has a low-level neural network associated with its motor drive. Initially the robot would issue detailed commands to the motor. If these detailed commands could be used to train the neural net then, over time, the robot could substitute higher level commands to the neural net rather than detailed commands to the motor drive.

Another example where context sensitive learning would be useful is when the robot is temporarily or permanently injured. New rules could be added, with the appropriate context, to compensate for the injury. If the injury is corrected then the context for the injury is removed and the new rules would be no longer effective.

### 4.2 Planning

A plan for our robot would be a contextual sequence. The sequence would be entered into the knowledge base as a set of rules, each sharing a common context (goal). When that goal is entered into the robot’s context, it would begin to follow the steps in the sequence. However as the context changes, other actions may take place. Local contextual

changes may cause the robot to temporarily deviate from the plan or even remove the goal from the context so that another plan could “kick” into effect.

Let’s use a “hungry lion” as an example. If our lion was implemented using a traditional goal oriented planner then the hungry lion might generate a plan by backtracking from the goal of eating. An abbreviated plan might be- get up, start wandering around, smell for prey, stalk prey, go for kill, then eat. If this plan is done at too high a level then important steps might be ignored. If the plan is generated at too low a level then it could well be too inflexible for real-world contingencies (e.g. the lion becomes thirsty wandering about the hot savanna). A larger question is how would the lion learn the plan, and how would the lion adapt the plan over time?

The contextual approach overcomes all of these problems. Instead of a master plan, the hungry lion has rules for what to do in various contexts when hungry (another context)-

```
Next Hungry LayingDown => Get up
Next Hungry Standing => Wander
Next Hungry SmellFood => LocateSource
Next Hungry SeePrey => Stalk
Next Hungry CloseToPrey => GoForKill
```

...

Here we have rules of what to do when hungry in various situations. The rules can be learned in any order. The rules can be specific or general. The rules can change over time as the lion’s environment changes. Other rules can be integrated to handle other circumstances-

```
Next Thirsty AtWater => Drink
```

If the lion, while hungry and looking for food, becomes thirsty then this rule would make it drink if it came across water.

### 4.3 Pretending

The ability to pretend gives our robot new capabilities in predicting what may happen in different scenarios (contexts). Multiple plans may be generated, each within a different context. The robot could also have a special context that disables its motor controls. Each of the plans to be evaluated could be “tried” in an imaginary context to see what the outcome might be.

For example, assume a robot is following a plan that involves crossing a body of water that is subject to tides. It has determined, based on its stored knowledge, that it will arrive at the water when the tides are low and hence will not have any problem crossing. The robot then detects something interesting. It determines that to investigate this new object of interest will delay it  $x$  hours. It could then shift its clock ahead (in a “pretend” context) and follow out its previous plan. If problems occur (such as it is no longer low tide) it could then decide whether or not to stick with its

original plan or go ahead and investigate the new object of interest.

## 5. The V4 Model

V4 is a model for the representation and manipulation of multidimensional, context sensitive data, rules and procedures. V4 has been implemented as a collection of C modules. It is currently running on a variety of operating systems and hardware platforms.

### 5.1 Dimension, Points, Bindings

V4 is based on a multidimensional data model. A **dimension**, in V4, represents a concept. **Points** along a dimension are instances of that concept. For example, points on the integer dimension represent each of the integers, points on the date dimension represent different dates. Dimensions can be for any concept so we can have dimensions for battery charge level,  $(x,y,z)$  coordinate systems, sensor inputs, etc. The syntax used by V4 is *dimension:point*. Examples could be Integer:123, Date:3-Jan-98, Color:Red, BatteryLevel:0.97, and Coordinates:<X:23.3 Y:48.8 Z:0.04>.

Information is stored within V4 as **bindings**. The intersection of dimensional points is bound to a value point. For instance

```
[Value:Sales Year:1997 Item:Widget] = Dollar:20000
```

denotes the binding of the intersection of three points (Value:Sales, Year:1997, and Item:-Widget) to the point Dollar:20000.

Bindings can be at specific points (as in the examples above) or over a range of points. For instance, factorial ( $n!$ ) can be defined with two bindings

```
[Func:Factorial Int:<=1] = 1
[Func:Factorial Int..] = Mult(Int* [Func:Factorial Minus(Int* 1)])
```

The first binding states that the factorial of all integers less than or equal to one is one. The second states that the factorial for all integers is defined as the product of that integer, (*dimension\** represents the current value in the context) times the factorial of that integer minus one. These two rules overlap, yet V4 will automatically choose the *best* rule for any given point on the integer dimension.

The above example referenced two of V4’s internal modules- Mult() and Minus(). The modules are invoked much like traditional procedures. V4 defines modules for enumerating through points of a dimension, performing mathematical, statistical, and financial calculations, and manipulating lists. Modules are also defined for manipulating the context and multidimensional space and can be used to construct new V4 rules and procedures at runtime.

The next example demonstrates how V4 modules can be used to manipulate points over dimensions to analyze data. In this example, the robot collects power usage by component over a time period. What we want is a summary showing the top (power usage) three components by system

on an hour by hour basis from data collected over a given time period. The first rule below defines the top  $n^{\text{th}}$  component by system and hour of day. It sorts all the components for that system by power usage for the time period and returns the  $n^{\text{th}}$  component.

```
Bind [Rank Hour.. System.. Int..]
List(Sort([Comps] Reverse::[WattHour],0) Nth::Int*)
```

The next rule below performs the steps for the analysis. The first Tally() enumerates through all of the components and makes a list of components by system. The second Tally() enumerates through all usage history points, selects those in the wanted time period, and sums watt-hour usage by component and time of day. The final Sort() sorts by system name and creates a table showing for each system, the top three components for each of the 24 hours in a day. EnumCL() takes two arguments, the first is a list of points (Hour:0..23 is a coerced into a list of hours from 0 to 23). The second argument is then evaluated in the context of each point in the first argument. The result is a list of the evaluated points (e.g. EnumCL(Int:1..10) @Mult(Int\* Int\*)) returns the list (1 4 9 16 ... 100)).

```
Bind [CompRank DateTime..]
Do{Tally(Comp.. (ListOf::Comp* By::Comp.System Bind::[Comps]))
  Tally(Usage.. If::In(Usage.DateTime DateTime*)
    (Sum::Usage.WattHour By::(Usage.Comp Usage.Hour)
      Bind::[WattHour]))
  Sort(System.. By::System.Name
    Do::(EchoT(System.Name
      EnumCL(Hour:0..23
        ([[Rank Int:1] Name] [[Rank Int:1] WattHour],0)))
      Enum(Int:2..3 If::DefQ([Rank Int*])
        @EchoT(" " " "
          EnumCL(Hour* ([[Rank] Id] [[Rank] WattHour],0))) )
```

The following example demonstrates several V4 features (note that dimension declarations have been left out to simplify the example).

The first three bindings define John's marital status at various stages of his life (Time:10, 20, 30)-

```
Bind [Status John Time:10] Single
Bind [Status John Time:20] Married
Bind [Status John Time:30] Divorced
```

The next three bindings relate is/will/was to various times. *Is* evaluates in current context ([R\*]), *Will* evaluates at Time:999999- the future. *Was* invokes a V4 module, IsctVals() which returns the value of [R\*], not at the current context but at the value associated with the prior context.

```
Bind [WhatS R?] [R*]
Bind [WhatWill R?] [R* | Time:999999]
Bind [WhatWas R?] IsctVals(@[R*] 2)
```

To evaluate the rules we must first define a time (Time:25) and then evaluate the status of John for the present, past, and future.

```
Context Add Time:25
[WhatIs Status John] => Married
[WhatWas Status John] => Single
[WhatWill Status John] => Divorced
```

## 6. AN EXAMPLE CREATURE

The following set of V4 rules demonstrates many of the concepts discussed in this paper. The "creature" implemented in these rules lives in a 100 by 100 units square area. The area has obstacles, food, water, and shelter. The creature spends its days meandering about. If it is hungry and smells food, it heads towards the food and eats it. If the creature, in its meanderings, detects a shelter, it remembers the position. As nighttime falls, the creature heads to a shelter or location (0,0) if no shelter has been found.

The main rule below steps the creature through 1000 time units for each of 10 days. The evaluation of [Senses] updates the context with information collected from the creature's senses. For instance, if the creature is "touching" food then the point Sfood:Yes is entered into the context, if the creature is not touching food then Sfood:No is entered. For each time unit the context point Hungry:number is incremented indicating that over time the creature becomes increasingly hungry. The evaluation of [Next] determines what the creature will do at the current time unit.

```
Bind [DemoCreature]
Enum{Day:1..10 @Enum{Time:1..1000 ([Senses] [Next] )
```

The rules below specify the meaning of [Next] in various contexts, i.e. what the creature will do in a variety of conditions. The default rule (first one below) states that the creature will reevaluate its goal coordinates (possibly changing them according to a random number generator) and then move one (x,y) unit towards the goal and set the current context to its new position. The second rule states that if the creature has any hunger (Hungry:>0) and is touching food (Sfood:Yes) then it "eats" by subtracting 100 from its Hunger context. This rule is effective until the creature has satisfied its hunger. The next rule reflects the cruelty of life. If the creature's hunger exceeds 1000 then it "dies" (Nop(0) is a V4 function that causes a failure). The fourth rule states that if the creature is touching shelter and the time is greater than 500 (nighttime), it is to do nothing (i.e. sleep). The last rule states that if the creature detects shelter and has not previously detected shelter then it remembers the position of the shelter.

```
Bind [Next] Do{[SetGoal] Context{[NewPosition]})
Bind [Next Hungry:>0 SFood:Yes]
  Context{Plus(Int:-100 Hungry*)}
Bind [Next Hungry:>1000] Nop(0)
Bind [Next SShelter:Yes Time:>500] Nop(1)
Bind [Next SShelter:Yes NPH:{undefined}]
  Context{MakeP(Dim:NPH N CPH.X CPH.Y)}
```

The next three rules set the creature's goal coordinates. The first default rule sets the goal to the position selected by the meandering random number generator. The second sets

the goal to the strongest smelling food source if the creature is hungry, if the smell exceeds a threshold, and the time of day is less than 700. The last rule sets the goal coordinates to shelter, if found, or (0,0). Note how the creature might be heading towards shelter after Time:500 yet divert to food if it is hungry.

```
Bind [Goal] GPH:=MPH*
Bind [Goal Hungry:>300 Smell:>100 Time:<700]
MakeP(Dim:GPH N Food.X Food.Y)
Bind [Goal Time:>500]
DefQ(NPH* @MakeP(Dim:GPH N 0 0))
```

The [SetGoal] rule, below, uses a random number generator to change the creature's goal point. If the goal point is changed then the context is altered. The dimension "Follow" is used when the creature is following the boundary of an obstacle. It is removed if a new goal point is set.

```
Bind [SetGoal]
Do(If((StatRan() > 0.9995 | ~DefQ(MPH*))
@Context(MakeP(Dim:MPH N StatRan(To::100)
StatRan(To::100))) Nop(1))
Context(TPH:=[Goal])
If((TPH.X <> GPH.X | TPH.Y <> GPH.Y)
@Do( Context(GPH:=[Goal])
Context(MakeP(Dim:Follow Special::Undefined)))
@Nop(1) }
```

There are other rules required to make this demonstration actually work such as how the senses update the context and how the creature actually moves. A complete listing can be obtained from the author.

## 7. Conclusions & Related Work

The use of contextual sequencing has been proposed as the primary model for the control of autonomous agents. Contextual sequencing offers many advantages

- It is a conceptually clean model. The basic ideas are simple yet the model can be used for both high and low level control.
- All contexts are potentially equal. This allows the knowledge base to utilize the entire context space without any artificial partitioning.
- This model permits extreme adaptability in differing contexts. New data and contexts may be dynamically added and removed.
- This model makes no distinction between data, rules, and procedures. An evaluation may result in a data element being returned or a lengthy procedure being executed. The model also allows for the self-construction of new data and procedures.

V4, a working version of a context sensitive, multidimensional evaluator, has been implemented. It runs on a variety of operating systems including Windows-95, Windows-NT, Digital OpenVMS, and several Unix dialects. V4 is supported on Intel, Digital Alpha, HP PA-Risc, and IBM RS6000 platforms.

## 7.1 Related Work

Other robot designs have been designed with a common area for all sensory input [4]. The author knows of no other research in this area where the idea of context is used as the primary determinant of "what to do next".

The ideas of Brooks [1] share a similar philosophy. His philosophy, in a very small nutshell, is that intelligent behavior, arises from the combined "efforts" of independent behavior modules. This bears some resemblance to the independent operation of each "next" step in the contextual model. The implementations of the contextual model and Brooks' are entirely different.

V4 shares many similar structures to other rule based systems. An excellent introduction and overview to these systems can be found in [3]. V4 differs from prior rule based or production systems through its use of a context.

There are many other models for controlling robots. The neural network design is currently being investigated by many others. The author does not want to imply that context should replace the neural network model but rather work with it as mentioned earlier in this paper.

## 8. REFERENCES

- [1] Brooks, Rodney A. "Intelligence Without Reason," *MIT A.I Memo No. 1293*, 1991.
- [2] Caudill, M. and Butler, C. *Naturally Intelligent Systems*. Cambridge: MIT Press, 1990.
- [3] Franklin, S. *Artificial Minds*. Cambridge: MIT Press, 1995.
- [4] Liscano, R. "Using a Blackboard to Integrate Multiple Activities and Achieve Strategic Reasoning for Mobile-Robot Navigation," *IEEE Expert*, Vol. 10 No. 2, pp. 24-36, 1995.
- [5] Schmajuk, Nestor A. "The Psychology of Robots," *Proceedings of the IEEE*, Vol. 84 No. 10, pp. 1553-1561, 1996.