A MISSILE CONTROL SYSTEM BASED ON AI IDEAS

Erik Skarman
SAAB Missiles
Linköping

Abstract

This paper describes a missile control system, BRAIN, which is based on Artificial Intelligence ideas. Some key properties of the system are:

- It is a three level architecture, where the mid level is rule based.
- The rule base is written in a language understandable to a human reader, and it looks like a consise specification of the missile system.
- The top level contains domain specific information in the form of algorithms for computations, that go beyond the expressional power of the rule base language.
- The mid level gets its "mental engergy" from alarm processes on the lowest level. These keep the mid level occupied with making the missile avoid a number of dangers.
- Other processes on the lowest level are dealing with control and perception.
- The number of state variables throughout the system is restricted as far as possible, so that state variables fall into one of three categories:
  - * Estimates of entities in the external world (which includes the missile hardware).
  - * Discrete obervations about the external world, like "It is snowing", or "The target is now visible".
  - * The intention occupying the mind of the system for the moment (usually the intention to avoid some danger of dangers).
- The processes on the three levels are so written, that they can be executed in parallell and in anarchy, and they communicate via a common bank of known state variables.
- The mid level processes are subprocesses to the low level processes, and thus hierarchially under the low level procedures. Hence, there is no centralized control.

General ideas

The state variables in a control system

Modern control theory teaches us to make control systems, in which the states of the controlled system are fed back to the inputs through constant gains (and not through any sort of filters). Put in another way: The dynamics required for good, stable behaviour of the closed loop system, is contained entirely in the controlled system itself. The control system doesn't need to contribute with any state variables.

A further stretch of this thought is to regard any state variable introduced in the control system as an unnecessary and undesirable artefact.

A justification for the judgement "undesirable" is that such a state variable contains information, and you can loose track of whether that information is relevant or not. This kind of problem can be seen, when some variables saturate, but other variables continue to charge up, and when a control system shifts its mode, but filters retain values that were only relevant in the previous mode.

This is even more true for the logical overhead that usually comes with a control system, to control modes etc. "Flags" and "Mode-words" and "Submode-words", tend to remember their lessons too long. The system designer looses track of all the information contained in these words, and how it affects the continued behaviour of the system.

The radical approach to this problem is to remove all state variables from the logical overhead too. The thought, that would make this possible, would be a generalization of the statement above, say: "All information needed to control a system, is contained in the controlled system itself", or slightly reformulated:

> "The situation of the controlled system, determines at any moment, which control actions ought to be taken".

We don't know whether this statement is true, but it seems to be true enough.

"Situations" and "problems"

In ordinary language, the word "situation" has a double meaning. Firstly it has a very neutral meaning, that being in a situation is being in a point in the state space. No situation is worse than any other. Secondly, it has a more loaded meaning as in sentences like "How did we get into this situation"? Here the word "situation" is more or less synonymous with "problem". Between the two one can imagine an interpretator, interpreting the situation - described to it very neutrally - as being "problematic" or not.

Our control system divides itself along the same lines.

It has one control level, which handles the situation very neutrally, by feeding the state variables back to the input, without any further "thought". Every point in the state space (every situation) is mapped onto a value sent to the input. We will call every such mapping, or "control law" or "feedback pattern" a stategy.

It has another control level, which handles situations interpreted as problems. It solves problems by taking actions. It maps every problem on a action:

problem -> action

The mapping from problem to action may be given to the system directly as a table. We call this table a rule base.

The rule above can be rewritten as

if problem then action

This looks more like a rule in the rule base of an expert system. The rules here share the property of the rules in an expert system, that every rule is independant of the other. All rules coexist in parallell. The rule base is merely a list of rules without any ordering or hierarchy or other structure.

Actions to be taken. Regression

Obviously one possible action to take is to tell the other control level to use another strategy. This is more universal than it may seem. If the input, that the controlled system expects from the control system, is an on/off-signal, then one possible strategy is to send the on-value irrespectively of the situation - or the off-value. Hence the rule base can describe control of on/off signals.

The second possible action is of a higher order. As result of the appearence of problem, the system may have to "withdraw to its rooms" and think the situation over. The result of this thought process can be a redefinition or reparametrization of some strategy, or a reformulation of some of the rules in the rule-base. We call this withdrawal process the regression process.

State estimation. Perception

As in ordinary state feedback control, the state of the controlled system is often not available directly through sensors, so it must be recontructed or estimated inside the control system. The Kalman filter is prototypical for systems, that can do this reconstruction. The Kalman filter is based on a model of the controlled system (or some part of it). The sensor measurement is compared to the expected measurement according to the model, and the difference, called the innovation, is used to update the model. Other, more or less Kalman-filter-like, estimation systems can be designed along the same lines. We call all these systems perception processes.

The concept of experience

The system may be said to solve its task by means of a amount of experience, which it may have gained by itself, or may have inherited from the designer, who in turn may have gained part of it from simulations with system prototypes. This experience contains mainly:

- Rules.
- Strategies (in terms of, for instance, gains in some control process).
- Models.

The regression process is solely occupied with updating this mass of experience, but it has no exclusive privilege to do that. The perception process may also update the models, if it contains an element of system identification in addition to its state estimation tasks. In this case the limit between state and experience becomes floating. Purely formally the experience information satisfies the requirements for being state information.

System architecture

We can now outline the system architecture for the entire system (see fig 1). The subsystems involved in the control process are called processes. We can identify the following process types:

- Control processes.
- Perception processes.
- Alarm processes.
- A decision process.
- A regression process.

The control process handles the direct mapping from the point in state space to the control system outputs (= the input of the controlled system). There is at most one control process for each output.

The perception process reconstructs the state vector from the input signals, using some models of the environment. There may be one perception process per model.

The alarm processes monitor that state vector or, in other words, the situation and interpret it as being problematic or not. There is at most one alarm process for each problem that is relevant.

The decision process has access to the rule base, and decides whether a problem alarmed for is to be considered, and which action is to be taken.

The regression process does the regression.

Together these proccesses form a three level architecture. The perception, control, and alarm processes are on the lowest level. They are run continously on some kind of cyclic basis. The decision process, which is rule based, is on the mid level. It is executed on demand from the alarm processes. The regression process is on the top level. It is executed on demand from the decision process. Large regres-

sion tasks can possibly be solved in a background environment on spare time from the other processes.

Communication within the system

The communication between these processes is also shown in fig 1. The state vector is accessable by virtually all processes, as via a common bus. The perception process writes its data on this bus.

The alarm processes all read this data. As a result of their interpretation they may send the names of their problems to the decision process.

The decision process has access to the rule base, and from this it makes a decision. The decision may be a new strategy, which is sent to the control process, or it may be a regression task, which is sent to the regression process.

The control process receives the strategy from the decision process, and forms a feedback pattern according to it. The state-variables then flow through this feedback pattern to the outputs.

Perception through alarm processes. Observations

Once we have freed ourselves from the Kalman filter scheme, the perception processes may very well estimate boolean variables, as well as real variables. But an alarm process together with the rule base also lends itself well for estimation of boolean variables. Once a problem has occurred, according to the alarm process, one may set a "the problem has occurred"-variable to true. When an alarm process has interpreted state data so that "it is snowing", one can make the observation that "it is snowing". This observation is then a part of the state information available in the system.

Rule base syntax and semantics

The rule base is an unstructured list of rules. Each rule has the form

```
problem -> action.
```

The action consists of different subactions in different channels. Each subaction is given with the syntax

```
CHANNEL:subaction;
```

and the action is an unstructured list of such subactions.

Each subaction may contain conditions, so that the subaction may be written as:

```
strat0,cond1 -> strat1,cond2 → strat2
```

This structure is searched from left to right, and the action used is the rightmost one, whose condition is satisfied. Conditions are quotings of possible observations. The condition is satisfied, if this quoted observation has actually been made.

Channels

There are channels for:

- Each control process (subaction: a strategy).
- The regression process (subaction: a regression task).
- Observations (subaction: an observation).
- Possibly individual parameters in the mass of experience (subactions like "up" and "down" or "reset").
- Possibly indivudial system outputs (subactions like "on" and "off").

All the subactions are texts, as clearly as possible expressing what the action means.

The observation-channel is actually divided into boxes. Every observation assigned to one box, overwrites previous observations assigned to that same box.

Some dynamical aspects

Alarm processes and subjunctive strategies

Alarm processes are not supposed to give alarm, when a disaster actually happens. It is supposed to give alarm, at the very latest, when the disaster is inevitable. Principally, however, no single disaster is ever inevitable before it has actually happened. If we may only consume limitless amounts of energy, pull as many g:s as we like etc, we can avoid any disaster in the very last moment before it happens. Alarm processes therefore occupy themselves with conflicts.

Suppose that we want to make a missile avoid hitting the ground. The way to do that is to pull up with some acceleration. If we assume constant acceleration during the pullup, it is straightforward to compute how big acceleration is required (for a given missile state, and a given terrain). A conflict arises when that acceleration reaches a level when the missile would stall. Before that moment, we need not worry about ground collision. There is a safe way to avoid it. After that moment we can only choose between ground collision and stall. The alarm, then, should be given at that very moment.

Note now, that while the alarm process analyses this pull up manoeuver, the missile does not at all do any pull up. The missile actually follows one strategy, while the alarm process makes computations as if it followed another. The alarm process works with a subjunctive strategy (with the word subjunctive used with the meaning it has in grammar).

Subjunctive strategies are used everywhere in the alarm processes. The system also contains a mechanism to actually control which subjunctive strategy is used in an alarm process. In the place of a strategy in a rule table, one

can put a more complex strategy, like this:

```
problem   CHAN:strat1/ALARM/strat2;
```

The system actually follows the strategy strat1, but the alarm process ALARM is told to analyze what would happen if strat2 were used instead.

Conflict areas

In the ground collision alarm that we just studied, there was a conflict in that the solution to one problem (not hitting the ground) led to another (stall). This conflict exists in some subset of the state space. The alarm process shall alarm when we reach the edge of such a subset. We cal call the subset

```
M(sol(P1),P2)
```

The missile moves in state space in a archipelago of such subsets. They have some interesting properties. One of them is that, for a given problem Q and a list of problems Pi (possibly including Q), all the sets.

```
M(sol(Q),Pi)
```

form a total order. It means that if you avoid the biggest of the sets you avoid them all.

Priority

Using the conflict sets as a tool, you can make a system, which avoids conflicts between different alarms. Two alarms requiring contrary actions in the same channel, would not appear simultaneously. However it happens that a missile is "born into" such a situation at the moment of launch. In that case one has to make a choice between the two actions. Hopefully one can neglect one of the alarms, and yet avoid a disaster. The problem may imply that a disaster happens with some probability. In that case, one could neglect the problem, and take a calculated risk. This would justify the introduction of a priority mechanism.

Syntactically it just means that a priority number is attached to a problem.

```
problem '7 -> action
```

(This problem has priority 7).

The simple rule for prioritation is: "If two alarms require actions in the same channel, that one wins which has the highest priority".

In a multi-channel system were an alarm may require actions in several channels, an extra rule is needed. We have chosen the following, out of many possible: "If an alarm does not win priority over its competitors, in all the channels, where it requires action, it is not considered at all".

This rule can be overruled with a mechanism that allows the designer to propose alternative actions for the same problem. Those alternative actions involve fewer channels, so that some of them may win priority in all its channels.

Once it is introduced, the priority mechanism can be used to simplify system design in many ways. For example one may have a "System being idle" problem meaning that the state is in no conflict set at all. This alarm may be always on, if it just has the lowest priority, so that all other alarms can defeat it when they occur.

Challenger torunaments and the concept of intention

An efficient way to implement the priority mechanism, is in the form of a challenger tournament.

Some alarm is the champion in that tournament. It is up to any new alarm to challenge that champion. The fight then is a simple comparision of priority numbers. When the champion alarm is withdrawn, it leaves a "blank" champion behind, which triggers all other alarms to try a new challenge. The principle is that an alarm tries a challenge, when it is new, or when the champion is new.

The "Champion" has the name of a problem, and that problem is the reason for all the actions taken by the system. In other words, it is the systems intention to solve that very problem.

This intention, in fact, is now a new state variable in the system. It is the only state, that is not an estimation of something in the outer world. It represents the systems own will.

It is an artefact of one so wants, but its a nice artefact, that one wouldn't easily loose track of. In fact it means, that it is easier now to keep track of the strategies. The strategies are no states, they are outputs. These outputs can be directly computed from the intention and the observations. This mapping from intention and observation to strategy is the rule base.

The dynamics between alarms and actions

Many alarm processes contain an element of prediction. Prediction has some special dynamical properties.

If some parameter in a strategy is changed, the system will react with a smooth transfer to a new form of trajectory, but a prediction of a point on that trajectory some long time ahead, will jump dicontinuosly to a new value. This may cause the alarm to fire, which may result in an equally discontinuos change in the parameter. There is then an algebraic loop in the system. The alarm will flip on and off at a high rate.

Sometimes this is a result of oversimplified prediction algorithms. Sometimes one must be careful in interpreting the alarms.

A system may contain an energy alarm process, which predicts the energy status till the end of mission. When this alarm process fires, the correct interpretation is not that we have a

lack of energy, but that we will have it if we follow the present plan. It is only natural, that the alarm will disappear, if we revise our plan. And if we, encouraged by the fact that the alarm has disappeard, reestablich the plan again, it is only natural that the alarm reappears. And so on.

Instead of revising the plan, we could make the observation that "Plan P causes energy problems", and that observation could then be a guide in the revision of the plan. That way we would at least not come back to plan P again.

System properties

Optimality?

The handling of the dynamics between the alarm and its action is probably the most difficult issue in the designing a BRAIN system.

Another difficult task is to make BRAIN systems which provide optimal solutions. The strategies used by the control processes are often optimal control laws, which optimize something under some few constraints, but the entire system is not very much interested in optimality. Its goal is "good enough", and its method to achieve that is "Solve the problems when we have them!".

The reason for this is that the BRAIN system is intended to manage complex control tasks with many constraints, where we believe it is too difficult to find optimal solutions. In principle, an optimal plan for solving a complex problem can always be made up, but doing it with reasonably small computer programs in reasonable time is another thing. And plans are no good anyway. They are not flexible enough. What we want is optimal control laws, and that is even more difficult.

It is very natural to define problems in terms of optimal solutions under constraints. But as a next step, it is also natural to put the question: "Is optimality really necessary?" "Or even relevant?" Often the problem can be reformulated into a "good enough"-one.

If this is so, the BRAIN-concept offers the designer a number of advantages.

System design with BRAIN

The rule base is written in a language understandable to a human reader, and is thus a good means of communication with the end user about what the system should do. In fact, the rule base gives a very concise and at the same time precise description of how the system should behave. The rule base language is a good specification language.

Hence it is a good start of system design, to make a preliminary rule base.

The rule base also forces the system designer to think about why something should happen in the system. The left hand member in a rule contains the argument for doing what is stated in the right hand member of the rule. This issue of argumentation is not at all so distinct, when the design is made with state transition diagrams.

When the designer writes his rule base, he has some idea about how the different problems, that he mentions in the rules can be detected. The natural next step is to specify these algorithms further. Other steps are to design the implementations of the strategies in the control process, and the tasks in the cognition process.

Once this work has resulted in a working prototype, all further work, like expansion, modification, adaption to changes in specifications or environment, maintenance etc, is done in the same way. New rules can be added and old ones can be removed. For new problems, new alarm processes are designed, and they can be added to the old ones without disturbing them. In the same way, new strategies can be added, and the regression process can be ordered to fullfill new tasks.

This modularity, and its consequence that program maintenance is made along the same lines as the original design, is perhaps one of the main advantages of the BRAIN concept.

Real time properties

Some of the control and perception processes usually have to be executed with a strictly fixed frequency. The alarm processes only need to be executed with a minimum frequency, usually lower than that of the fastest control processes. Hence it is usually suitable to have at least two program cycles, one of which can actually be a background cycle without any fixed frequency.

This means that programs interrupt each other, and that always means hazards. In this case we believe that the standardization of the process types and of the comunication between them, minimizes these hazards.

An alarm process reads state data computed by a perception process in another cycle. As long as we feel happy when the alarm process gets the latest state data available at every moment, everything is OK. Sometimes it may be important that certain data items in a package of data are from the same time. This is not too common, though, when the state vector mainly contains continuous type data about the situation. We have consistently avoided logical types of data in the state vector.

The only type of data traffic in the opposite direction is transfer of strategies to the control processes. The new strategies are delivered as quickly as possible, and the control processes will follow the new strategies as soon as they are delivered. The only possible risk with this traffic, is when an interrupt occurs the moment when the strategy is transfered. In that case mixtures between the old and the new strategy may be visible to the control process. It is relatively easy to

build a protection mechanism against this.

Reusability

We have built systems according to this concept mainly for two types of missiles. One is a ground attack missile flying over hilly terrain towards a single known target, and equipped with an optical seeker. The other is an anti-ship missile manoeuvering in archipelagoes, or flyig over open sea against possibly large groups of potential targets, and equipped with an active radar seeker. In spite of the large difference in task, environment and expected behaviour for the two missiles, over 50% of the software is common between the two programs. In spite of the differences, so much is still in common between the two guidance tasks, and the BRAIN concept helps us discover this commonality.

This also means that when some new system feature has been tested and approved for one of the missiles, it can readily be taken over by the other one, if it is a relevant and desirable system feature for that other missile too.

Conclusions

We have found that by generalizing the ideas from state feedback in ordinary linear control theory, mainly by being very stringent about introducing states in the system, we can give a scheme for designing control systems with very good properties:

- The scheme can be used for solving a large variety of control tasks, probably even very complex ones.
- There is a natural path to follow, when designing a system, a path that starts with the overall requirements on the system.
- Communication with the end user of the system is simplified by the relative readability of the rule base language used.
- The designer is seeing the arguments he used for the design explicitly in the design.
- Maintenance and modification of the system is made along the same lines as the original design.
- The system is very robust to modifications, e.g. when new functionality is added.
- Real time implementations are relatively easy to make safe.
- Software for one project can be reused in anogher to a large extent.