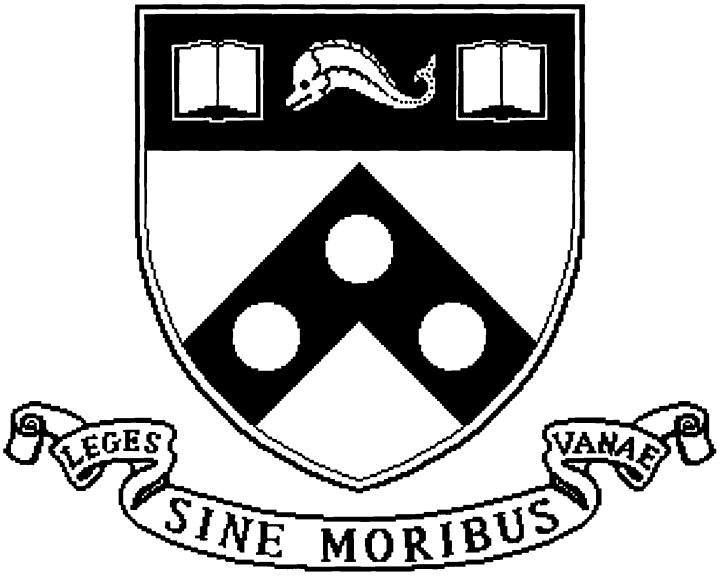
Sensor Planning with Bayesian Decision Analysis

MS-CIS-95-14

Steen Kristensen [1]



University of Pennsylvania
School of Engineering and Applied Science
Computer and Information Science Department
Philadelphia, PA 19104-6389

1995

[1]This work was done at the GRASP Laboratory, University of Pennsylvania, USA

Sensor Planning with Bayesian Decision Analysis

Steen Kristensen[1]
Laboratory of Image Analysis
Aalborg University
Fredrik Bajers Vej 7D
DK-9220 Aalborg E
Denmark
E-mail: sk@vision.auc.dk
URL: http://www.vision.auc.dk/~sk

March 24, 1995

[1]This work was done at the GRASP Laboratory, University of Pennsylvania, USA.

Preface

This technical report is the documentation of the work I did during my 6 months visit from August 1st 1994 to February 1st 1995 at the GRASP Laboratory, University of Pennsylvania, Philadelphia, USA.

The report has evolved from being working notes into what is presented here. I hope that this does not decrease the readability of the report.

I would like to thank Dr. M. Mintz, Dr. R. Bajcsy, and Dr. H. I. Christensen for their valuable discussions and R. Mandelbaum for providing much of the sonar sensing stuff. Of course any shortcomings are mine and not theirs.

Steen Kristensen

Aalborg, March 1995

Contents

Chapter 1

Introduction

To make an autonomous mobile robot navigate in natural environments it is necessary to equip it with sensors allowing it to sense the surroundings to detect obstacles and freespace. Furthermore, it should be capable of estimating its own position by localizing known objects so that it does not get lost.

Since a single sensor can not efficiently fulfill all these demands it is of interest to investigate how to make a number of sensors cooperate to gather the information necessary for an autonomous robot to safely and accurately complete its tasks. The process of deciding which sensors to use for which purposes is called *sensor planning.*

The nature of the sensor planning problem depends on the architecture of the system considered. In this work we consider a system designed after the *purposive paradigm* where a number of specialised functional modules, also called *purposive modules*, are employed to solve the robot navigation task. We will define a purposive module as a self–contained entity capable of doing its own sensing, processing, and actuating—all specialised for a single purpose.

Sensor planning has become increasingly relevant with the advent of the purposive paradigm and active perception [1]. This is due to the fact that in a system with several purposive modules there has to be a mechanism for deciding which purposive modules to invoke for solving a given task. The use of active sensing strategies at the same time means that many sensors/actuators[1] are un–sharable between modules since each module configures the sensing system for its own purposes. Therefore, there has to be a sensor planner to schedule the sensing resources and a planner for deciding which purposive modules to invoke. In the purposive paradigm this is the same thing, however, since a purposive module with no sensors allocated can not do any useful work and thus sensor control and control of which purposive modules to run reduce to the same thing. The central issue for the sensor planning is thus to decide what modules to grant the control of what sensors in what sequence, in order to achieve the best overall performance of the autonomous mobile robot system. Therefore, what we would like to have is a system like the one depicted in figure 1.1.

The sensor planning problem has been approached with various techniques, e.g., has fuzzy set theory been applied to build sensor preference graphs [4], probabilistic methods have been used

[1]When dealing with active sensing the border between a sensor and an actuator is blurred and thus we will from now on use the term "sensor" in the meaning "sensor/actuator".

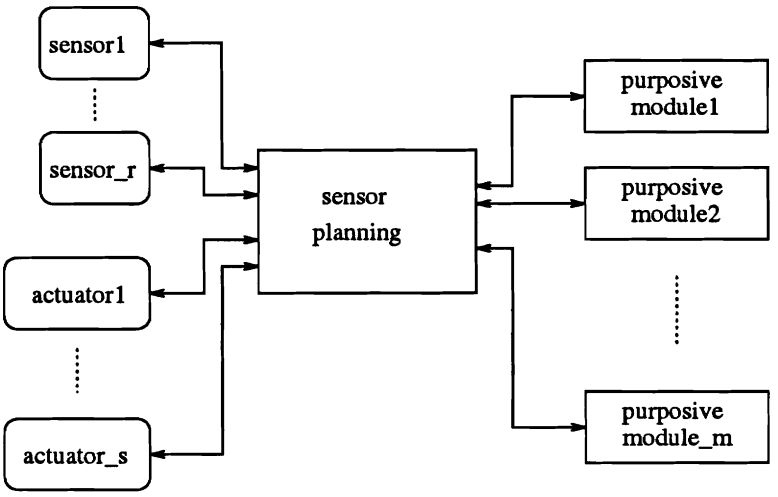


Figure 1.1: *Illustration of a purposive system with handling of sensor/actuator contention. Some sensors might be accessed by several modules at the same time, and some may only be used by a single (or no) module.*

to determine sensing actions [3] [5], and traditional planning techniques have been applied to control sensing to bring parameter errors inside the acceptable bounds [7]. However, in the literature there seems to be slightly different opinions about what the term "sensor planning" covers. Some authors (e.g., [7] and [13]) define the problem as selecting the optimal sensor parameter values given one or more sensors. This, of course, is primarily of interest with respect to active sensing. Another part of the sensor planning problem is not *how* to use the sensors but *what* sensors to use (e.g., [4]). This is also called the *sensor selection problem.* It is that part of the problem that is addressed in this work. Because of the system architecture with purposive modules we also indirectly consider the *integration problem* described as "to provide an effective model of assembling a system's modules together and incorporating the information provided by each module into the operation of the whole system." [2]. Thus, what is described in this work as the sensor planning framework could also be thought of as an integration framework.

It has been chosen to use Bayesian Decision Theory for the sensor planning since it is a framework that allows for reasoning about uncertainties which is considered crucial in real world applications. Furthermore, this approach allows for a more modular, and thus scalable, approach than, e.g., traditional rule–based approaches. Bayesian Decision Theory has traditionally been used in economics and in that terminology the purposive modules are competing for the resources in a cost/benefit manner according to probabilistic models. How this is done is the topic of the remainder of this report where in chapter 2 the basics of using Bayesian Decision Analysis (BDA) are introduced. This can be skipped by readers familiar with the theory. In chapter 3 it is described how BDA is used for sensor planning. How this is then used to control the robot system is outlined in chapter 4. The aspects of using time with the framework are presented in chapter 5. In chapter 6 a description language for describing sensors, tasks, and purposive modules is developed. After this, experimental results are presented in chapter 7 and the results are discussed and conclusions drawn in chapter 8. Further research directions are finally given in chapter 9.

Chapter 2

Bayesian Decision Analysis

In this chapter the basics of BDA will be explained and it will be argued why BDA is a good choice for sensor planning. The contents of this chapter are primarily based on the books by Pearl ([11]), Jensen ([6]), and Raiffa ([12]).

Basically it can be said that a Bayesian framework has been chosen because in an autonomous mobile robotics environment things are in general uncertain: sensory inputs are noisy, state variables do only reflect the real state of the world to a given degree, and furthermore an action performed by the robotic system can have different outcomes from time to time (and I will not even mention the funding situation here). In that context, Bayesian Decision Theory allows for "reasoning" about uncertainties, or rather, it provides a framework for systematically dealing with uncertainties in decision problems. This is because in a Bayesian framework knowledge is not represented by, e.g., production rules but rather as beliefs and conditional probabilities. Conditional probabilities is a way to express "probabilistic causality" or production rules with associated probability measures. For example, the production rule:

$$if\ A\ and\ B\ then\ C \tag{2.1}$$

states the knowledge that event A and event B causes C as a hard fact. However the conditional probability:

$$P(C|A,B) = x \tag{2.2}$$

states that event A and event B causes C with probability x. Bayesian calculus then provides us with means to generalize this to the case where A and B are only true with certainty y and z. This means that Bayesian Decision Theory allows us to express both causal relations and reason with uncertainty, or soft evidence.

Before going into details about the benefits BDA can provide for the sensor planning problem it will be appropriate with a couple of simple examples to illustrate what decision analysis and especially Bayesian Decision Analysis is all about.

With due respect to the historic heritage of the theory we will consider examples from the realm of gambling. This has the advantage of making the terminology more natural since it largely reflects the game theory origin of decision analysis.

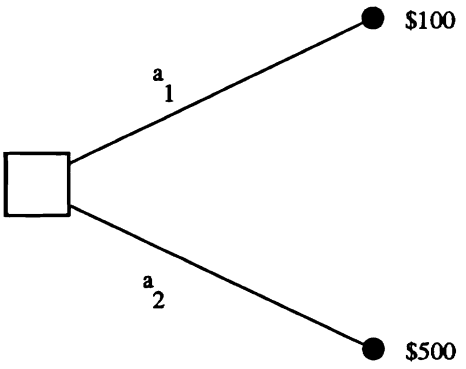


Figure 2.1: *Graphical illustration of NN's choices. The box is called a "decision node" and the whole structure is called as "decision tree." This is a common way to illustrate decision problems. In decision trees there is an implicit time axis, or "causality axis", going from left to right. In this case it means that first NN has to make a decision and then she can receive the prize.*

2.1 Actions and Lotteries

Whenever a decision problem exists it reflects the fact that some agent has an opportunity to make a choice or to perform an *action* (note that to do nothing is also some kind of action). If there is no possibility of choosing between actions, there is no decision problem. In the sensor planning case the action is to grant a functional module control over a sensor.

Now we will consider an example where the agent, NN, has two possible actions a_1 and a_2. Choosing a_1 will for sure earn NN \$100 while choosing a_2 will earn NN \$500. Assuming that NN will always choose the action that potentially will earn her the largest profit[1] NN will have no trouble deciding for action a_2. NN's situation is illustrated in figure 2.1.

It is said that NN has the choice between the *lottery* L_1 with *consequence* $C_1 = \$100$ and the lottery L_2 with consequence $C_2 = \$500$. When consequences are concrete material values such as money, they are also called *payoffs* or *prizes*. In this case the term "lottery" seems a bit awkward since NN was able to know the outcomes of her actions for sure. This, however, is only seldom the case in real world situations and as mentioned it is basically never the case in mobile robot navigation. Let us therefore complicate the situation a bit and let NN have the choice between two lotteries, L_1 and L_2, where L_1 with probability 0.5 earns NN \$100 and with probability 0.5 earns her \$20. Likewise, in lottery L_2 NN has a probability of 0.4 of winning \$500 and probability 0.6 of winning \$10. What action should NN now choose? Intuition alone says that NN should still choose action a_2, i.e., to participate in lottery L_2 since the probability of winning the "big prize" is almost as good as winning the significantly smaller prizes. NN's new situation is illustrated in figure 2.2.

[1] What Raiffa calls an EMV'er, EMV = Expected Monetary Value.

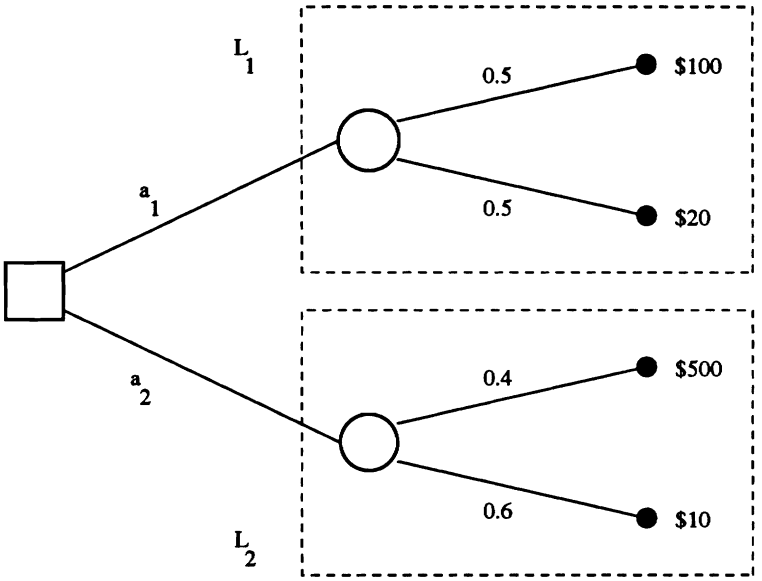


Figure 2.2: *Illustration of NN's choice between lotteries L_1 and L_2. The circle is called a "chance node" since Chance decides which "path" will be taken.*

2.2 Expected Payoffs

However, we (or NN) can not always trust intuition, especially if the situations are more complex or if the differences between choices is seemingly small[2]. But why is it that NN's "intuition" tells her to pick L_2? It is because NN *expects* to win the largest prize by choosing L_2. So what exactly should NN expect to win by participating in L_1 and L_2 respectively? If we assume that NN holds *all* the M lottery tickets for L_1 she would *surely* win $M \times 0.5 \times \$100 + M \times 0.5 \times \$20$. So when NN only participates once (or equivalently holds only one M'th of the lottery tickets) she must expect to win one M'th of the prize or $0.5 \times \$100 + 0.5 \times \$20 = 60\$$. When participating in L_2 the expected prize would likewise be $0.4 \times \$500 + 0.6 \times \$10 = \$206$. It is evident that NN can never win \$60 nor \$206 since these prizes do simply not exist, and thus it may seem artificial that this anyway is what NN should expect to win. In the long run these prizes will be the best guess for the average win, however, if NN consistently chooses a_1 or a_2. Therefore, to optimize her expected payoff NN should always chose option a_2. And this is what Bayesian Decision Theory is all about: To choose the action that in the long run will optimize the payoff!

To show how to apply this framework to sensor planning it is in place to formalize the maths and to generalize to the universal case where the consequences are not monetary values and where probabilities are not known exactly.

[2]We shall remember that NN is an EMV'er and thus *has* to figure out which lottery is the best. Therefore, what Minsky calls Fredkin's Paradox is not a solution to our problem. Fredkin's Paradox says that "The more alike two alternatives are the more time we spend on choosing between them. But that is paradoxical since the more alike two alternatives are, the less it matters which one we choose."

2.3 Utility

In robot navigation and many other aspects of life there is not a concrete monetary value associated with different choices. Rather there are some more or less abstract consequences. In order to be able to rank consequences and to enter very different kinds of consequences into the same framework, BDA operates with the notion of *utility* where it is assumed that all consequences including monetary payoffs can be "converted" into this common "currency" called utility. This is admittedly a sensitive part of Bayesian Decision Theory since how can, e.g., medical cost and the cost of a life be evaluated against each other in terms of utility? Luckily the consequences we are about to face are not as dramatically different as this but still we must remember that BDA only makes sense to the degree the ingredients make sense and to that end the assertion of utilities is a very important aspect. However, if we for now assume that we can assert these utilities, let the utility of an action, a, given the state of nature, z, be denoted $U(a, z)$. If in the example illustrated in figure 2.2 the upper path of each lottery is denoted z_1 and the lower z_2 then the utilities would be: $U(a_1, z_1) = U(\$100)$, $U(a_1, z_2) = U(\$20)$, $U(a_2, z_1) = U(\$500)$, and $U(a_2, z_2) = U(\$10)$. If the probabilities for the events z similarly are denoted $P(z_1|a_1), \ldots, P(z_2|a_2)$ the *expected utility* of selecting action a_i can be written as:

$$EU(a_i) = \sum_{j=1}^{M} P(z_j|a_i)U(a_i, z_j) \tag{2.3}$$

where M is the number of possible outcomes of the chance experiment, in the given example $M = 2$. Now it is possible to formulate Bayes Decision Rule as:

$$a_{BDR} = \arg \max_{a_i=a_1}^{a_N} EU(a_i) \tag{2.4}$$

where the associated expected utility is

$$EU_{BDR} = \max_{i=1}^{N} EU(a_i) = EU(a_{BDR}) \tag{2.5}$$

From equation 2.3 it can be seen that the utilities $U(a_i, z_j)$ can be scaled according to an affine transformation without this affecting the ranking of the corresponding actions, a_i, i.e.:

$$U'(a_i, z_j) = A \cdot U(a_i, z_j) + B, \ A > 0 \tag{2.6}$$

So far, the situation has been simplified by the fact that the utilities have been assigned directly to a choice. But often the decision maker has to make several choices before the payoff can be collected. Returning to our example this corresponds to the situation where NN does not win a cash prize in the first lottery but rather wins the right to chose which of a new set of lotteries to participate in etc. until a final lottery actually has a cash prize. The situation is illustrated in figure 2.3 where one of the consequences of lottery L_1 is to choose between two other lotteries to participate in. These lotteries in turn have a monetary prize.

In BDA this situation is handled in a recursive manner—when the expected utility of action a_1 is calculated, there has to be prizes (or rather utilities) on the leaves of the chance nodes. This is not the case in the problem in figure 2.3 but it is recognized that instead of the "missing prize" there is another decision problem of the type depicted in figure 2.2. This means that we

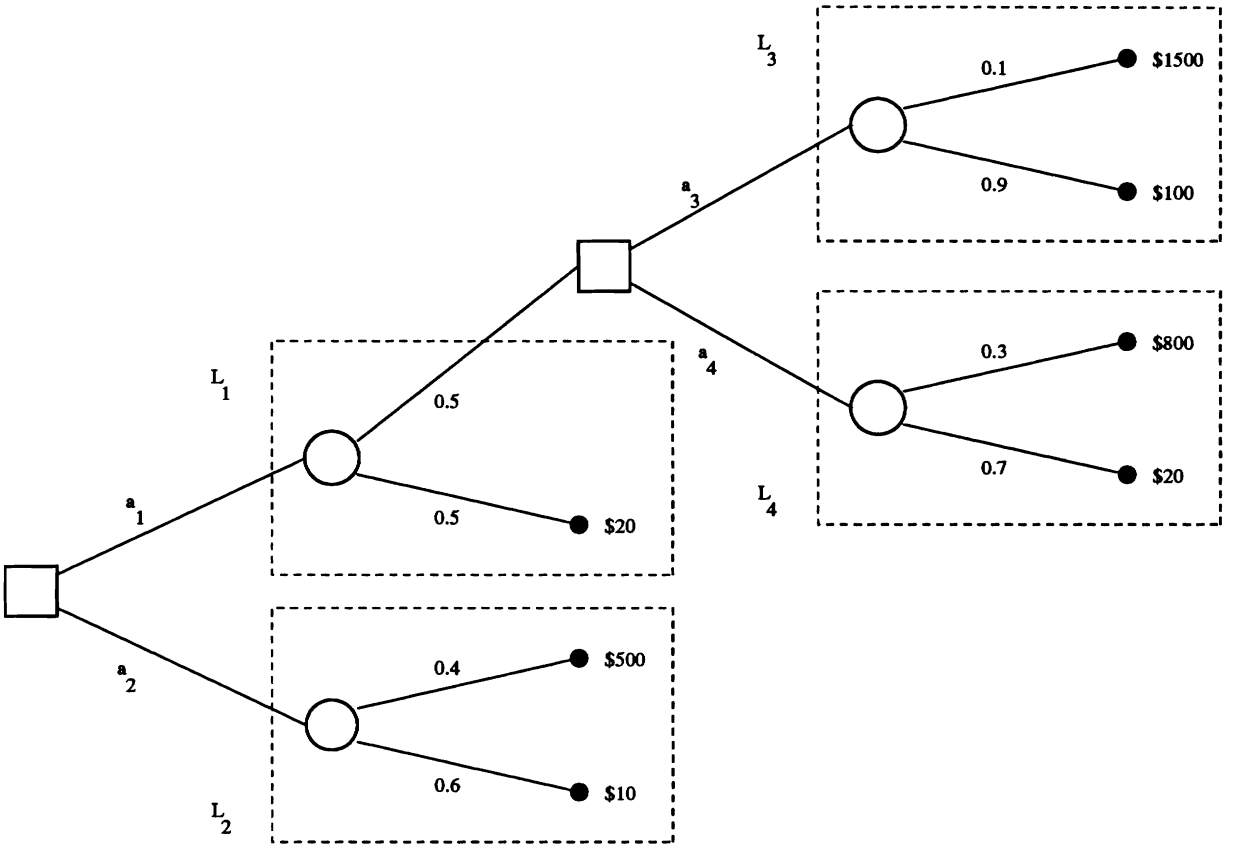


Figure 2.3: *Illustration of case where one or more choices must be made before a prize can be collected.*

can first solve this smaller problem and substitute this sub–tree with its expected utility. It is seen that the expected utility of participating in L_3 is $0.1 \times \$1500 + 0.9 \times \$100 = \$240$ while it for L_4 is $0.3 \times \$800 + 0.7 \times \$20 = \$254$. Thus, Bayes Decision Rules says that *if* NN should ever find herself in the situation where she can choose between a_3 and a_4 she should choose a_4. This means that the whole sub–tree is expected to be worth \$254. Now with this clear the total problem is reduced to the type of the problem from figure 2.2 and it can be shown that NN should still choose a_2. This recursive process is called *folding back* (the decision tree).

2.4 Costs

The process of folding back is relevant to the sensor planning problem since the task of the mobile robot is normally not the sensing itself but this is rather a means for completing some overall task. Thus, the robot should only be "payed" at the completion of a task and all the sensing decisions made underway should merely ensure that the utility at task completion has been optimized.

This in a natural way brings up the topic of another ingredient in the BDA framework: If the robot only has one task to complete there should only be one final utility, so how does it matter what sensing decisions to make underway? The answer is, of course, that not all sensor actions

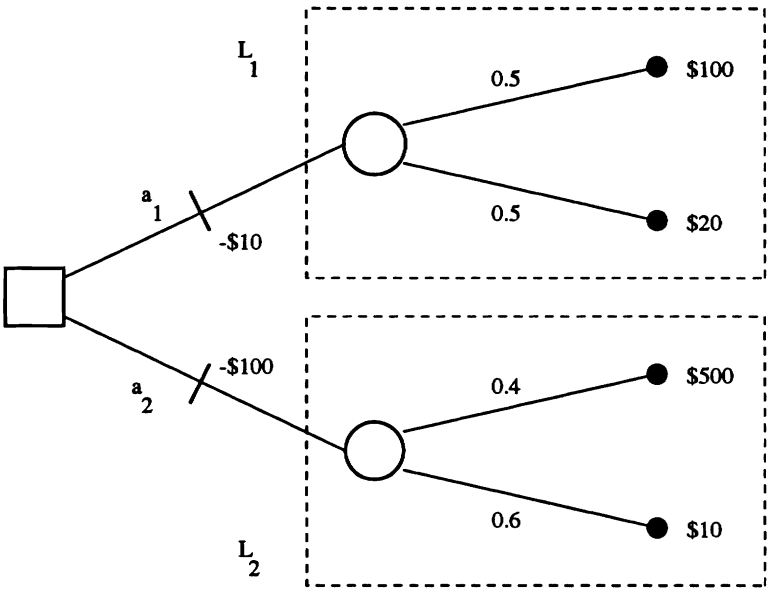


Figure 2.4: *Illustration of NN's choice between lotteries L_1 and L_2 when the cost of participating is \$10 and \$100, respectively.*

cost the same. The cost can be measured as time, power consumption, computational burden, and many other things or some combination hereof. Thus, to maximize the expected utility the sensor planning has to ensure that a large amount of utility (or *utils*, which is the "money" of BDA) is not spent on sensing in the process.

Returning to NN and her decision problems, the introduction of costs may seem even more adequate since participating in a lottery (especially ones as attractive as NN's) normally requires the purchase of a lottery ticket. In other words, if you play you pay. The concept of cost is however quite easily incorporated into the BDA framework. To illustrate how, reconsider the simpler example from figure 2.2 in the case where a ticket for lottery L_1 costs \$10 but a ticket for L_2 costs \$100. This is normally illustrated as shown in figure 2.4.

One way to handle the costs is to simply propagate them out to the utilities at the leafs of the decision tree and here subtract them from the prizes. It is noted that the costs and the prizes must be in the same "currency" to make any sense in the BDA framework, and thus costs are in general considered as being negative utilities. The case where the costs in the problem in figure 2.4 have been propagated is shown in figure 2.5.

It is recognised that the situation in figure 2.5 is now again identical to the situation in figure 2.2 and therefore equations 2.4 and 2.5 can be applied.

2.5 The Value of Information

So far it has been assumed that the outcome probabilities, $P(z_i)$, have been known exactly a priori to the decision problem. This is very valuable information for the decision maker to possess. Unfortunately, it is only rarely the case that the decision maker actually has this information, or rather, that the decision maker has exact a priori information. As a matter

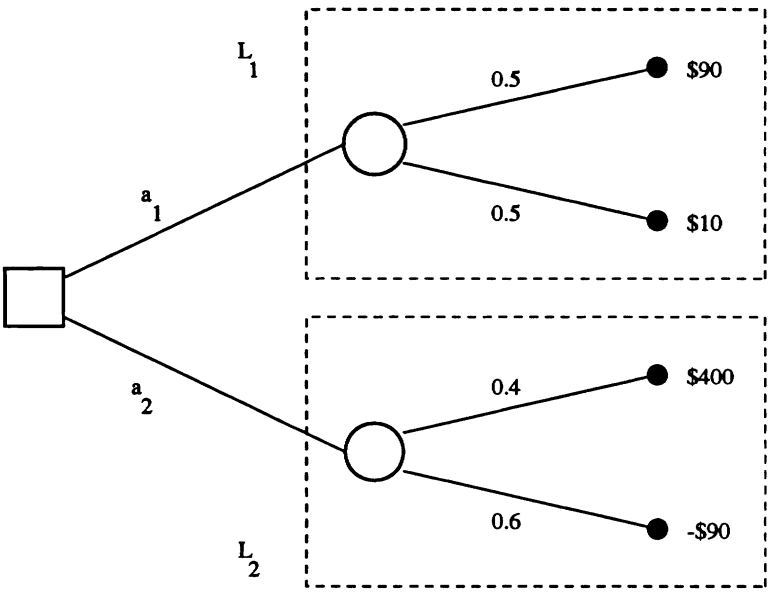


Figure 2.5: *Illustration of NN's choice between lotteries L_1 and L_2 where the costs of participating ($10 and $100, respectively) have been propagated to the leaf nodes.*

of fact, many decision problems and in particular the sensor planning problem is all about to decide whether or not the current information about the state of affairs should be improved by means of buying information from information sources (as, e.g., sensors).

Relating this to NN's decision problem it corresponds to NN not exactly knowing the probabilities of the outcomes in the two lotteries. We can, e.g., assume that lottery L_1 is an official state lottery and thus the probabilities are here known and exact, but the more lucrative lottery L_2 is run by an "underground" organisation and thus the odds are somewhat dynamic. Normally the probabilities are as on figure 2.2 but on some occasions when the head of the organisation (let us for short call this person the Godfather) is in a bad mood the probability of winning the $500 prize falls to 0. NN knows from experience that the Godfather is in a bad mood about twice a week, or in 30% of the time. But luckily, NN knows a person in the Godfather's family that is willing to sell information about the Godfather's current mood. Now, the question is: Should NN buy this information and how much should NN be willing to pay? The decision problem is illustrated in figure 2.6.

From figure 2.6 it can be seen that the value of the information about the Godfather's mood is worth $39.2 since the a_{buy}–path has an expected payoff of $89.2 while the a_{not}–path has an expected payoff of $50. In other words, if the family member will sell the information about the Godfather's mood for less than $39.2 then NN should buy it, otherwise NN should not buy it and rather play the state lottery. The reason why the state lottery is now more lucrative than the "family" lottery given that NN has no information about the Godfather's mood is that with no information NN must expect to win the $400 (i.e. event z_1 comes out) with a probability of $P(z_1|badmood)P(badmood) + P(z_1|goodmood)P(goodmood) = 0.0 \times 0.3 + 0.4 \times 0.7 = 0.28$. Similarly, $P(z_2) = 0.72$. This means that when the cost of participating and the moody nature of the Godfather is accounted for there is a 28% chance of winning $400 instead of a 40% chance of winning $500 and thus the state lottery is actually the better choice.

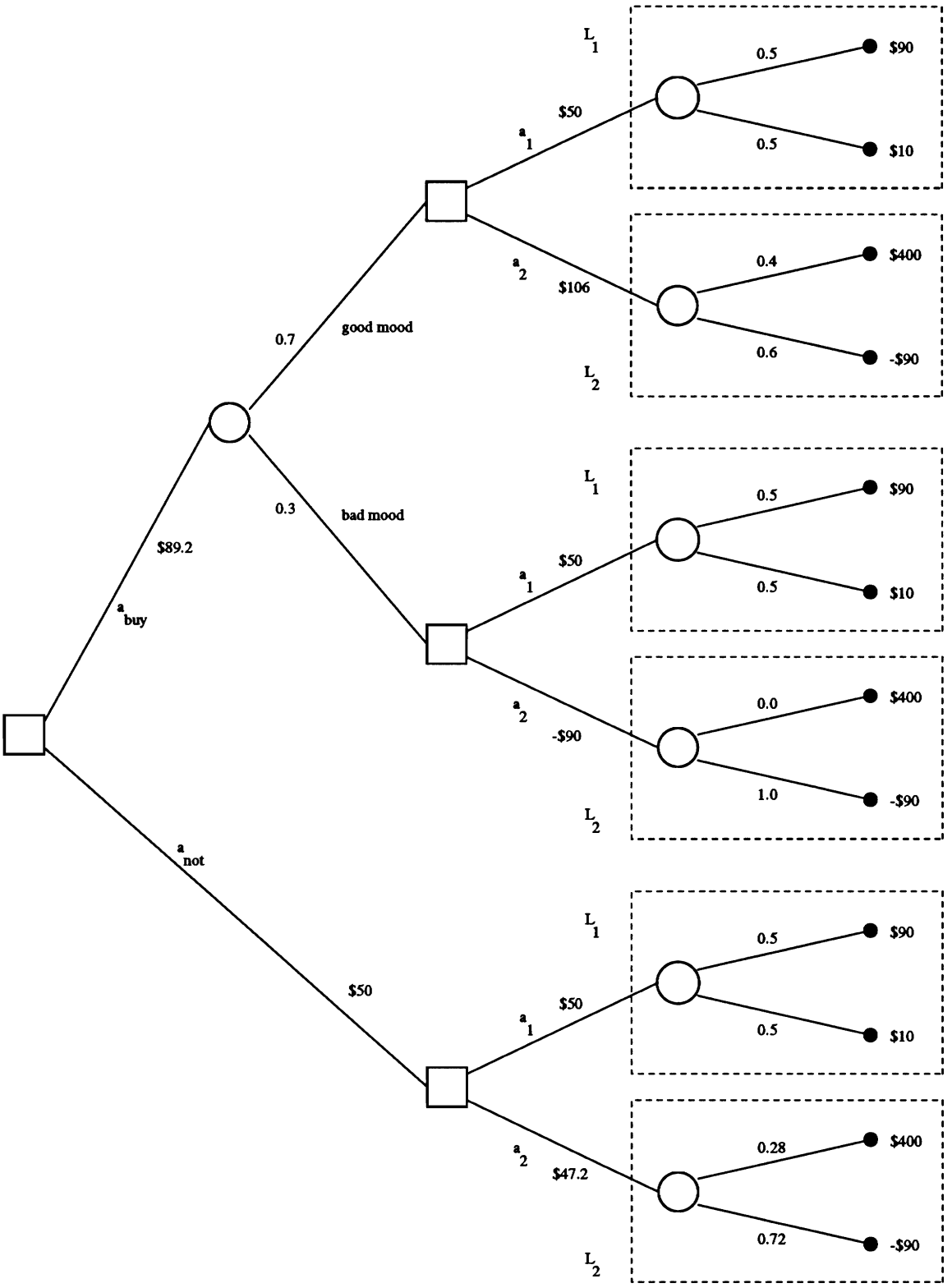


Figure 2.6: *Illustration of NN's choice between buying or not buying information before deciding what lottery to play. The prices for participating but not the price of the extra information have been propagated to the leaf nodes. The expected utility of each choice is written at each choice branch.*

	good	bad
happy	0.8	0.1
angry	0.2	0.9

Table 2.1: *Table of conditional probabilities for Mr. Loose's assertion of the Godfather's mood, P(loose|mood). Mr. Loose either reports "happy" or "angry."*

The information buying scenario is fundamental to the sensor planning problem in that the vehicle's navigation system can assume that the world is in a certain state and act accordingly. But since the state of the world can not always be known, or rather, can never be known (in the case of a dynamic world) information can be "bought" by employing sensing activities. How much the sensing must cost depends on how much use the information provided by the sensor is of with respect to the current task. The usefulness in general depends on two things: How much information the system already has about the state that the sensor can provide information about and how trustworthy the sensor is. The first criterion just means that if a sensor can only provide information that is already available it should not be employed (which it will not be in the BDA framework). The latter criterion reflects the fact that it is only seldom possible to get perfect information from a sensor but that we should be willing to pay more for near perfect information than for very uncertain information. This point is so central to BDA that it will be discussed in detail below and we will for the last time return to our lottery example.

2.6 The Value of Uncertain Information

Suppose now that NN knows two persons in the Godfather's family—the one (Mr. Perfect) that is very close to the Godfather and thus has perfect information and another person (Mr. Loose) more loosely "related" to the Godfather. This person sometimes misjudges the mood of the Godfather but for the same reason he is willing to sell his information quite cheap. From experience NN knows that Loose asserts the Godfather's mood with the probabilities listed in table 2.1.

Now to get the more complicated problem into the basic framework of equation 2.4 and the averaging out and folding back scheme, all that is necessary is to perform a couple of calculations. First, we would like to know how often Loose will come out with the statements *happy* and *angry*, respectively. This can be determined by *marginalisation*:

$$\begin{aligned} P(happy) &= P(happy|good\ mood)P(good\ mood) + P(happy|bad\ mood)P(bad\ mood) \\ &= 0.8 \times 0.7 + 0.1 \times 0.3 = 0.59 \\ P(angry) &= P(angry|good\ mood)P(good\ mood) + P(angry|bad\ mood)P(bad\ mood) \\ &= 0.2 \times 0.7 + 0.9 \times 0.3 = 0.41 \end{aligned}$$

What we furthermore need to know is what the odds for winning the $400 are when Loose reports *happy* and *angry*, respectively. In other words, we want the probabilities $P(mood|loose)$. And this is where Bayes Rule finally enters Bayesian Decision Theory![3]. Bayes Rule is an inversion

[3]Actually the theory is named after Bayes Decision Rule and not what is commonly known as Bayes Rule. However, it would have been peculiar if Bayes Rule had not entered the framework somewhere.

	happy	angry
good	0.95	0.34
bad	0.05	0.66

Table 2.2: *Table of conditional probabilities for Godfather's mood given Mr. Loose's report,* $P(mood|loose)$.

formula stating that:

$$P(B|A) = \frac{P(A|B) \times P(B)}{P(A)} \tag{2.7}$$

or,

$$P(mood|loose) = \frac{P(loose|mood) \times P(loose)}{P(mood)} \tag{2.8}$$

Since the three quantities on the right–hand side of equation 2.7 are known it is straightforward to obtain the result in table 2.2.

This information can finally be used to calculate the odds for winning the \$400 given Mr. Loose's information. This final process is called *averaging out* and the total evaluation of a decision tree is normally called *averaging out and folding back.*

$$\begin{aligned}
P(z_1|happy) &= P(z_1|good) \times P(good|happy) + P(z_1|bad) \times P(bad|happy) \\
&= 0.4 \times 0.95 + 0.0 \times 0.05 = 0.38 \\
P(z_2|happy) &= P(z_2|good) \times P(good|happy) + P(z_2|bad) \times P(bad|happy) \\
&= 0.6 \times 0.95 + 1.0 \times 0.05 = 0.62 \\
P(z_1|angry) &= P(z_1|good) \times P(good|angry) + P(z_1|bad) \times P(bad|angry) \\
&= 0.4 \times 0.34 + 0.0 \times 0.66 = 0.136 \\
P(z_2|angry) &= P(z_2|good) \times P(good|angry) + P(z_2|bad) \times P(bad|angry) \\
&= 0.6 \times 0.34 + 1.0 \times 0.66 = 0.864
\end{aligned}$$

The numbers are entered in figure 2.7 where it can be seen that the expected payoff given Loose's information is \$77.26. This means that NN should prefer to buy the information from Mr. Loose if he is $\$89.2 - \$77.26 = \$11.94$ cheaper than Mr. Perfect. However, NN should still prefer the state lottery if Mr. Loose charges more than \$27.26 for his information.

Although the cases for purchase of perfect and uncertain information have been treated separately it should be clear that the former case is merely a special case of the latter which is thus generally applicable.

One more time the importance of the process of deciding whether and where to buy information about the state of affairs should be stressed. This is at the very core of sensor planning and together with the other techniques described, Bayesian Decision Analysis is believed to provide a sound theoretical basis for solving the sensor planning problem. In order to get the concepts more clear, the subjects treated in this chapter are summarized below.

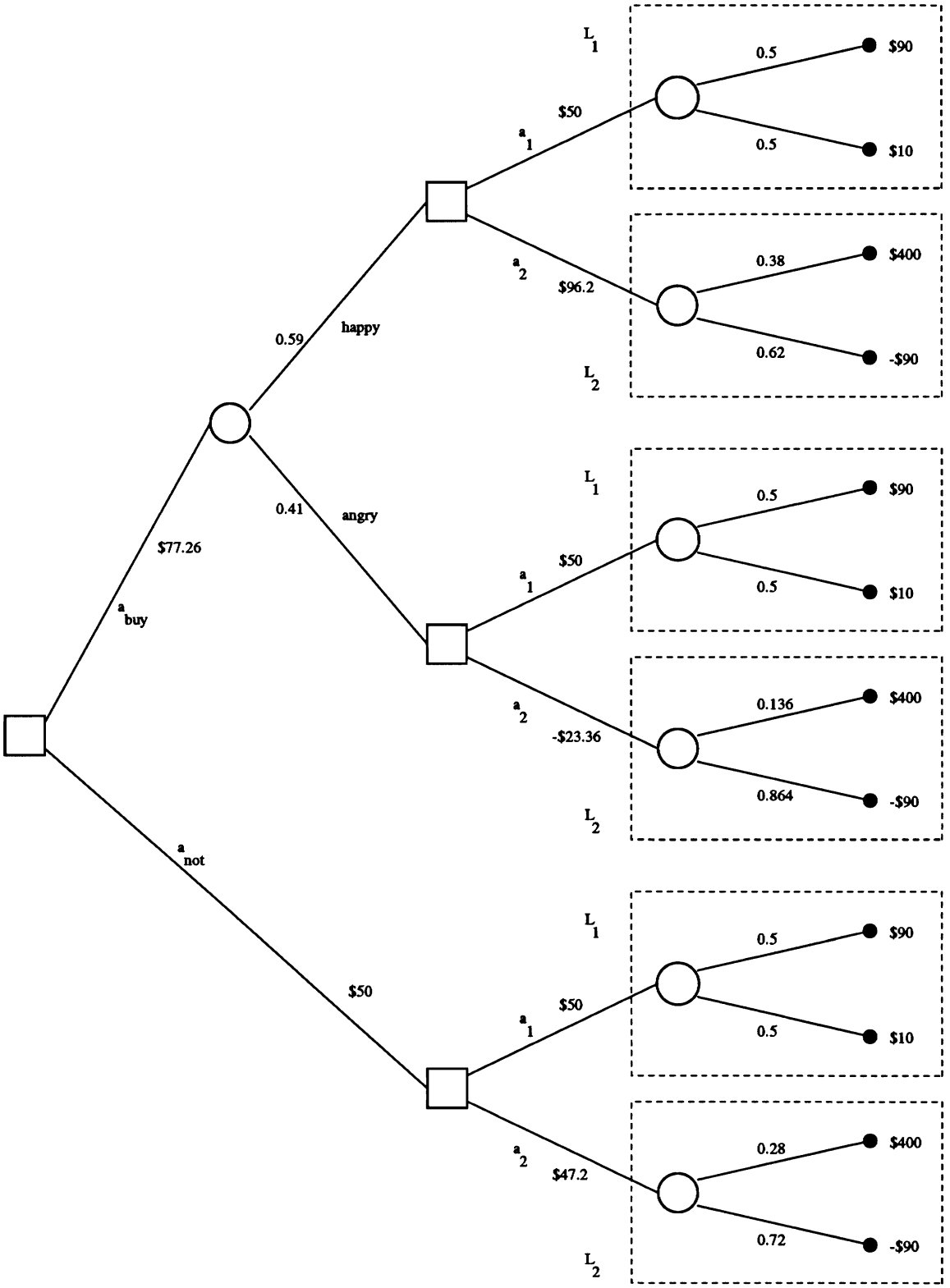


Figure 2.7: *Illustration of NN's choice between buying or not buying imperfect information before deciding what lottery to play. The prices for participating but not the price of the extra information have been propagated to the leaf nodes. The expected utility of each choice is written at each choice branch.*

2.7 Summary

The fundamental idea behind BDA is to select the action that in the long run will optimize the expected utility. This is expressed in equations 2.4 and 2.5 where the expected utility of an action is found by equation 2.3. Where no utilities can directly be associated with a chance node, the averaging out and folding back process is used. As the name indicates, this process consists of two steps. First the averaging (out) is performed. This is done by starting from the root decision node and propagating conditional probabilities outward towards the leaf nodes in the decision tree. The propagation property is due to the fact that decision trees are closely related to general Bayesian Networks and thus the probability at one chance node only depends on the parent node and a conditional probability distribution relating the parent node to the child node. The precise structure of the propagation depends somewhat on the structure of the decision problem and the exact mechanism will therefore not be explained until in the next chapter where the problem structure is determined. However, the propagation in general includes averaging, marginalisation, and conditional inversion using Bayes Rule. When the averaging out is completed, possible costs are propagated to the leaf nodes and the folding back is initiated. The folding back is basically just applying Bayes Decision Rule (equation 2.4) recursively until the root decision node is reached. The action found to be optimal at the root node is then the action to perform.

As it has hopefully been made clear, BDA is merely a mechanism for dealing with uncertainties in decision problems and it is the things that go into the framework that decide if the mechanically made decisions make sense. Therefore, let us review the ingredients that have been used:

Utilities, $U(a, z)$ This is a measure of desirability of the consequences of a given action. Utility is used because different and often abstract consequences have to be ranked and compared and thus a common measure of value is needed. Utility can be positive as well as negative corresponding to rewards and expenses, respectively. The utility "currency" is called *utils*. Since utility is the "driving force" of the whole decision mechanism it is very important that these utilities are somehow sensible in a *relative manner*[4]. Whether this is possible to obtain is an open question, but methods, however subjective, have been developed to alleviate the problems. Raiffa [12] provides a very nice discussion of this subject.

Conditional probabilities, $P(X|Z)$ These conditional probability distributions (CPD's) are really models relating cause and effect in a probabilistic manner. Since the very end–product of the averaging out is probabilities for various effects (or events) it is also important that these models reflect the world they are supposed to portray since otherwise the decision mechanism will be "misled." Depending on the type of cause–effect relation modeled by the conditional probabilities it can be either a quite easy or a formidable difficult task to establish these probability distributions. Unfortunately, it is also quite difficult to analytically determine the effects of modeling errors on the decision process, which is mainly due to the highly nonlinear nature of the problem.

A priori probabilities, $P(Z)$ This is a kind of model knowledge similar to the CPD's. The a priori knowledge is normally "injected" into the mechanism to get the averaging going.

[4]The absolute value of utilities is not important since this is an abstraction, and it can be shown that all utilities can be scaled by an affine transformation (equation 2.6) without this affecting the decision problem [12], [3].

Again the sensitivity of the decision analysis towards the a priori probabilities is hard to analyze analytically.

Costs, $C(a)$ Costs really just enter BDA as negative utilities and thus what applies to utilities applies to costs. The reason for listing costs separately is that they are often mentally considered to be different from the end consequence type of utilities. This is probably because all costs often originate from the same "parameter space" and thus only one transformation from this space to utility space is needed.

The basics of the BDA framework should now be established, but although it all seems quite mechanical the framework still has to be fitted to the specific problem, which in this case is the sensor planning problem. This is the topic of the next chapter.

Chapter 3

The Sensor Planning Problem Structure

In this chapter it is described how it has been chosen to structure the sensor planning problem in terms of Bayesian Decision Theory. Ideally, we wish to bend the Bayesian framework to fit the sensor planning problem. Before even attempting this there is, however, an important danger to be aware of. First of all, the title of this chapter is misleading—there is no structure of the sensor planning problem, there are at best a number of structures depending on how the problem is formulated. This is a common pitfall to plunge into since it is natural to formulate the problem in terms of the available technology. So as Descartes and Leibniz thought of animals and human beings as clockworks and as cognitive scientists think of them as information processing systems so could we think of the sensor planning problem as a Bayesian Decision Theory problem. There is, however, nothing intrinsic in the sensor planning problem that makes it a BDT problem and thus we will first try to formulate the problem more generally and then try to make it fit the BDT framework. This will hopefully make more explicit the compromises made to make the problem fit the technology. The ideal was however to make the framework fit the problem but since this is obviously not possible all we can do is to shape the technology to impose less constraints on the original problem. Clearly, this will also be a matter of compromises since in an embodied system as a robotic system, there are also implementational constraints to be met. This will be dealt with in detail in chapter 4.

3.1 Problem Statement Transformation

The initial formulation of the sensor planning problem will be as formulated in the Introduction:

> **Problem Statement no. 1:** To decide what module to grant the control over what sensors in what sequence in order to achieve the best overall performance of the navigation system.

Although the problem may be stated even more generally this formulation serves its purpose in that it is related to the general navigation system architecture considered here without being expressed in terms of the methods used to solve the problem. The only term that could

seem somewhat BDA–ish is the term "best overall performance" which could be interpreted as "maximum utility." Clearly, the choice of applying BDA to the sensor problem by definition assures that the system will perform so as to achieve maximum utility. But this alone indicates that "maximum utility" is not the same as "best overall performance" since it is easy to build a system using BDA that does nothing at all! Actually, as indicated in the previous chapter, there are many constituents of the BDA framework that have to be determined carefully before it even has a chance of doing something sensible.

So what does the term "best overall performance" then mean? We will here try to make the point that when interpreted as a global characteristic it does not mean anything at all in a traditional scientific positivistic sense but that it none the less could be a meaningful definition in a local sense. With "local" is understood "with respect to the current task." This local property is also indirectly reflected in the problem statement since it is understood that the sensor planning has to have a finite time horizon, or "sequence", to plan over (refer to section 4.1 for a further discussion of tasks and sub–tasks).

To be able to talk about "optimal performance" in a strict scientific manner it is necessary to define this with respect to some kind of criterion that in the end can be evaluated quantitatively. Such typical criteria are Least Squared Error, Least Mean Error, minimum time consumption, minimum power consumption, etc. All these metrics however depend on a (theoretical) reference that the actual results can be compared against since it is not possible to talk about "error" unless there is something "perfect" to deviate from. This again calls for a complete, analytical model of the problem from which such references can be derived. In other words, to scientifically talk about "best overall performance" of the sensor planner we must be able to define a reference performance which requires an analytical model of the problem domain. In a dynamic, seemingly undeterministic, real world situation such a model is not possible to construct, however, and thus it is not possible to define a criteria for optimal performance. It is for the same reason that it makes no sense to talk about humans as, e.g., "optimal drivers" or "optimal cooks"[1]. The point is that in order to evaluate actions that demand some kind of intelligence it takes an intelligent observer (what intelligence is, and whether the observer has to be more intelligent than the observed we will not even try to discuss here). As the goal of this project is to make a sensor planner that would enable the vehicle to behave (apparently or emergently) intelligent in a real world environment, the final judgment must be left up to intelligent creatures, in this case humans.

What can be concluded from all this is that when seen as a global characteristic the term "best overall performance" must be understood in a strictly subjective way (some would even claim that it can therefore never be achieved but this in itself is no reason not to pose it as a goal). Furthermore, it has become clear that although the Bayesian Decision Theoretic framework ensures maximized expected utility this is not the same as ensuring optimal behaviour of the sensor planning. However, if we only consider a limited sequence of sensory actions, as, e.g., related to a sub–task, then it might be possible to define meaningful task criteria to strive for. Such criteria could be minimum time, minimum power consumption, or maximum distance to objects (e.g., in the case of path planning). The way to enforce such criteria is then to engineer the utilities to reflect the various preferences. This implicitly means that different sub–tasks can have different task criteria which is natural and as a matter of fact it is considered an

[1]It can however be shown that humans are *not* optimal decision makers in a Bayesian sense where test subjects have been given decision problems that can be modeled analytically. All this indicates, though, is that this is not the kind of problems people normally deal with.

important aspect of animal intelligence to be able to change strategies when required. If the task criteria are then selected "intelligently" the hope is that the system in a global sense will exhibit intelligent characteristics. With this in mind we can formulate compromise no. 1:

> **Compromise no. 1:** By casting the sensor planning problem into a Bayesian Decision Theoretic framework local optimality is encoded as maximum expected utility.

This leads to a new problem statement:

> **Problem Statement no. 2:** To decide what module to grant control over what sensors in what (finite) sequence in order to achieve maximum expected utility for the navigation system.

It is implicit in the above formulation that it is assumed that all consequences and costs can be converted into utility. The assumption can be made true be simply doing it which however has the penalty of removing consequences and costs another step from the "physical" world into the more abstract utility space. Furthermore, the utilities must reflect the task criteria. This is considered as compromise no. 2:

> **Compromise no. 2:** By transforming all consequences and costs into utility, these concepts are made increasingly abstract and thus more likely not to reflect the real world situation and task criteria. This also makes the (subjective) analysis of the system harder.

From the discussion of the Bayesian framework it became clear that it is necessary to have a probabilistic model of the functional modules from which the framework can make its averaging out and folding back which eventually leads to the decision. Creating these models is another abstraction with the errors and constraints this kind of modeling imposes. One of these constraints (which is serious enough to get its own "compromise") is that the state space is discretized, i.e., there can only be a finite number of states for each random variable[2]. In practice, the number of states even has to be relatively small to keep the problem computationally tractable. This is a case where we are clearly shaping our problem after the technology and thus we have two more compromises:

> **Compromise no. 3:** By modeling all functional modules statistically with conditional probability tables errors are introduced as a consequence of simplifications and modeling errors. Again, this also makes the (subjective) analysis of the system harder.

> **Compromise no. 4:** The fact that all random variables have to be discretized into a relatively few states imposes considerable constraints on the generality and precision of the models and thus the entire problem.

[2]Recent research in Bayesian Networks [10] may render this constraint obsolete by allowing continuous random variables, but the exact result and implications are not yet known to me.

As a remark to Compromise no. 4 it can be said that the discretization limits what in other frameworks is the called the *expressional power*.

We are now ready to yet another re–formulation of the sensor planning problem:

> **Problem Statement no. 3:** To decide from (discrete) statistical module models which of the corresponding functional modules to grant control over what sensors in what (finite) sequence in order to achieve maximum expected utility for the navigation system.

With this problem statement we are now ready to consider how to shape the Bayesian framework to match the problem.

3.2 Bayesian Problem Formulation

In a Bayesian framework, problem statement no. 3 means that a complete decision tree encompassing the entire sensor planning problem for the current task should be created and evaluated to find the best sequence of sensing actions. Besides being computationally intractable this approach also has the problem of being self–contradictory as argued by Pearl [11]:

> "In fact, once we have the facility to run a complete analysis of the problem, the need to evaluate and rank information sources vanishes, since the optimal strategies found already contain a precise prescription as to which information to consult and when."

Also, the concept of planning to the completion of the task is contradictory to the whole idea of a responsive and flexible system that can cope with unforeseen events in the environment.

Another reason for not planning far ahead is that for each level in a decision tree the probabilities are propagated through, the effects of modeling errors accumulate and will tend to render the distributions diffuse and thus the final decision meaningless with respect to the physical state of affairs. It would somehow correspond to making a 1–year weather forecast.

The conclusion is that planning ahead to the task completion is *not* a feasible approach and we will now argue why to take the direct opposite approach—only to plan one step ahead, which is also known as *myopic* decision making. Although planning to the task completion is not really an option, we will still state the refraining from doing it as a compromise:

> **Compromise no. 5:** By using myopic decision making the potential sub–optimality of not planning to the task completion is accepted as preferred to the intractability of planning far ahead.

The decision of choosing myopic decision making is so important that it requires a new problem formulation:

> **Problem Statement no. 4:** To decide from (discrete) statistical module models which of the corresponding functional modules to grant control over what sensors at this instance in time in order to achieve maximum expected utility for the navigation system.

It should be clear that planning as little ahead as possible keeps the complexity of the problem to a minimum and thus reduces the computational burden. This is important since we would like the computational demands of the sensor planning to be small compared to the demands of the sensory actions so that the planning does not have to account for the cost of its own planning.

It may be argued that planning "a little" ahead (we will call this *shallow planning*) would not increase the computational burden to a prohibitive level. The step from not planning ahead to shallow planning is however critical in another sense. When not planning ahead, Nature will provide the "feedback" in terms of the consequences of the performed action. This is then automatically accounted for in the next "iteration" when the next decision is to be made. This is not the case even for shallow planning where a casual model relating actions to expected effects (on the entire system, including state variables) must be provided. In general, sensing actions are not believed to have any effects on the world[3] but this is not entirely true since active sensing, e.g., can affect the state variables for the vehicle (position, rotation, and associated uncertainties) which again can influence subsequent actions. Modeling these "side effects" is believed to be extremely tedious and error prone and choosing myopic decision making is thus to accept the consequences of ignoring these effects rather than to spend resources trying to model and incorporate them into the decision making process. We will state this a the next compromise:

> **Compromise no. 6:** By using myopic decision making the error of not accounting for sensory action effects on the world state is preferred over the burden of modeling and accounting for these effects.

Having settled on myopic decision making it is now possible to formalize the concrete BDA framework.

3.3 Myopic Decision Making

The standard structure of the myopic decision problem is illustrated in figure 3.1 [11].

The root decision node is called *sensory action* since this is the basic decision task of the sensor planner as defined by problem statement no. 4. If there are m modules that request the service of a sensor the planner can choose between $m+1$ actions, namely to grant the sensor to one of the m modules (normally called the *informed* case) or not to grant the sensor to any module (the *un–informed* case). The latter would be the case if the maximum expected utility of granting

[3]Being a Dane, I should of course know better than to neglect the result of my famous country–man Niels Bohr, who stated that "It is not possible to observe a system without affecting it." We will however trust the fact that quantum effects are ignoreable in this context.

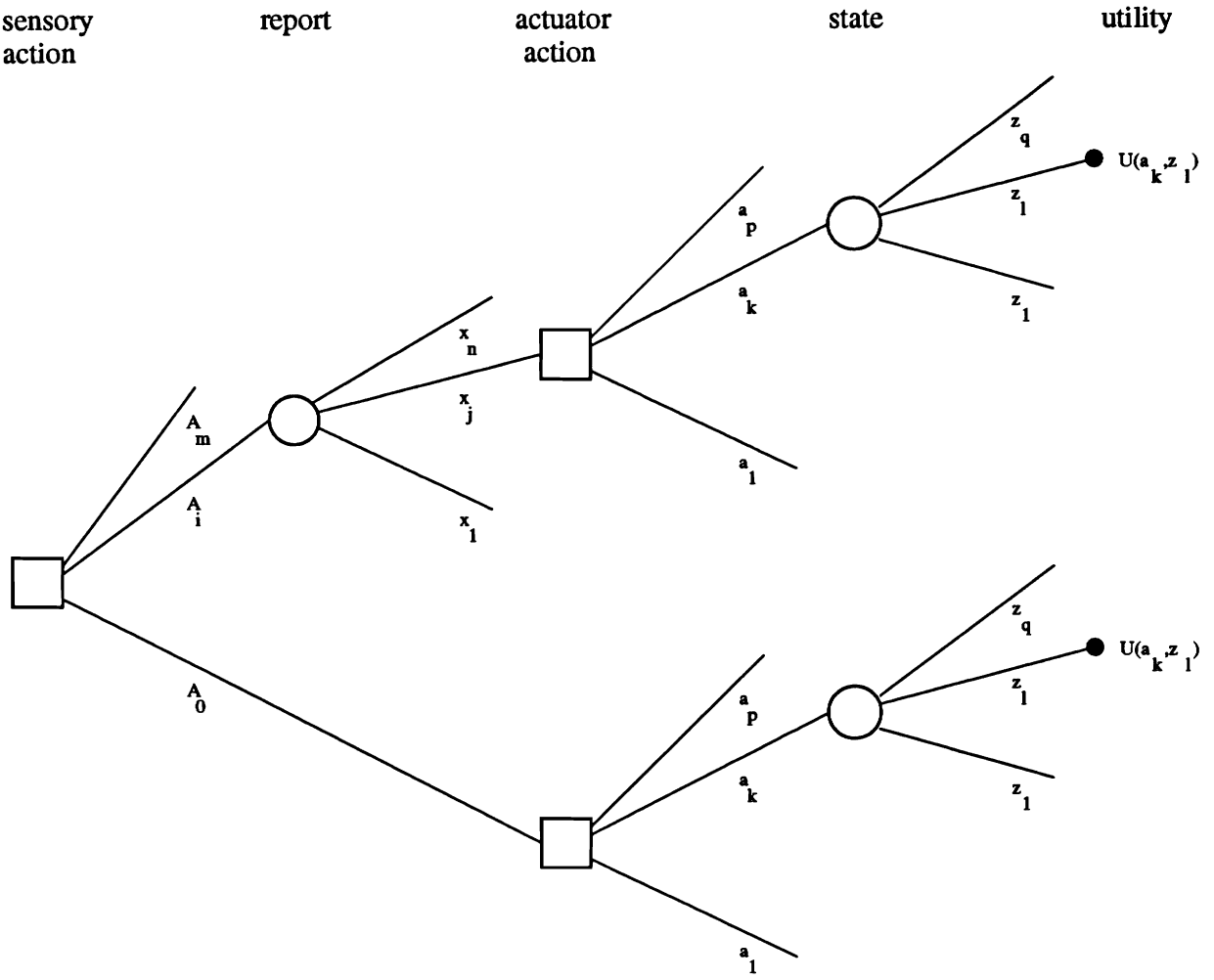


Figure 3.1: *The standard structure of the myopic decision problem, adapted from [11].*

the sensor to a module is smaller than not granting it at all which again would mean that the cost of sensing would not make up for the expected value of the information.

The *report* chance node represents the random outcome, x_j, of the sensing action, A_i. It is noted that this chance node is absent in the un–informed case since this implies that no sensing is performed. There is a slight notational abuse here since it is clear that the outcome of the chance node in general will depend on which action A_i has been chosen and thus a more proper name for the outcome would be $x_{i,j}$ to indicate the dependency of the random variable X on A_i.

The second decision node, called *actuator action* is the node representing the final consequence type of action that will lead to the utility payoff. There could be cases where this action is not an actuator action in the traditional sense, but the term has been chosen to distinguish between sensory actions that represent information gathering (and thus expenses) and final, somehow productive actions marking the completion of a task and thus a utility "income." This set of actions will generally be independent of the previous sensory actions, and thus a_k, $k \in [1,p]$ is the same set for all choices, A_i.

The *state* chance node represents the world state, Z, that the world assumes "after" the actuator action has been chosen. It is of course only in the representation that the world state is set after the actuator action has been chosen, but the reason for representing it this way is that it is only at this time that the world sate matters and is finally asserted. For example, if the chosen actuator action, a_i, is to drive through a door then the state of that door (open or closed) is certainly going to be asserted.

The utilities, $U(a_k, z_l)$, in figure 3.1 denote the payoff for completing a certain task. The utility is dependent on which action is chosen to complete the task, *and* the state of the world. For example, if it is chosen to drive through a door and the door is open this should result in a high utility. However, if the door turns out to be closed this should result in a low utility.

Seen as a whole, the myopic decision scheme can been interpreted as "given that at most one sensory action can be performed before task completion, which one should this be in order to maximize the expected utility?." Interpreted according to the implicit time axis (from left to right) the decision scheme means "first we have to select a purposive module to grant the sensor, then we get a measurement. On basis of this we select an action which will ultimately reveal the state of the world and decide what utility is earned." Of course only the granting of the sensor should really be performed, since this automatically generates the report and the rest of the decision tree (see figure 3.1) merely serves as the problem context or the driving force of the system. It is thus also clear that more than one sensory action can be performed and that the actual actuator actions should only be performed when the A_0 action is found to be optimal, i.e., when continued sensing can not increase the expected utility. This means that as soon as a sensory action (other than A_0) has been chosen, the system "loops back" to the root decision node and re–evaluates the situation to see what is now the optimal choice. This corresponds to using a local search strategy like, e.g., gradient descent, in traditional optimization problems. This means that the same problems with local minima are present, which is what is meant with the term "potential sub–optimality" in compromise no. 5.

Now, we will summarize the math of the myopic approach.

The expected utility, $EU(a_k|x_j)$, of a state chance node is:

$$EU(a_k|x_j) = \sum_{l=1}^{q} P(z_l|x_j)U(a_k, z_l) \tag{3.1}$$

The state probability, $P(z_l|x_j)$, is unconditioned on the sensor action A_i which is a result of the fact that we assume the sensory actions have no effect on the state of the world (the sensory actions are *non–intervening*).

From equation 3.1 and Bayes Decision Rule (equation 2.5) the expected utility of receiving report x_j, $EU(x_j)$, can be found as:

$$EU(x_j) = \max_a EU(a_k|x_j) = \max_a \sum_{l=1}^{q} P(z_l|x_j)U(a_k, z_l) \tag{3.2}$$

It is assumed that the cost of performing actuator action a_k is included in (i.e., has been subtracted from) $EU(a_k|x_j)$.

The expected utility of action $A_i, i \in [1, m]$ is thus:

$$EU(A_i) = \sum_{j=1}^{n} P(x_j)EU(x_j) - C(A_i) = \sum_{j=1}^{n} P(x_j)\left[\max_a \sum_{l=1}^{q} P_i(z_l|x_j)U(a_k, z_l)\right] - C(A_i) \quad (3.3)$$

where $C(A_i)$ is the cost of performing sensory action A_i. The subscript, i, on $P_i(z_l|x_j)$ indicates that when evaluating action A_i the evaluation should be made according to the conditional probabilities for module M_i. The expected utility of action A_0 is:

$$EU(A_0) = \max_a \sum_{l=1}^{q} P(z_l)U(a_k, z_l) \quad (3.4)$$

The sensory action (or rather, the sensor "granting") with highest expected utility, $EU(A_i), i \in [0, m]$, is then performed and the system "loops back" to the root decision node, unless in the un–informed case, where the best action, a_i, is actually performed which completes the current task. The associated control aspects will be described further in chapter 4.

Here we can now make a concrete list of the ingredients that go into the decision framework.

Utilities, $U(a_k, z_l)$ These utilities reflect what it is worth taking action a_k when the world is in state z_l. The cost of performing actuator action a_k must be included.

Conditional probabilities, $P_i(z_l|x_j)$ These conditional probabilities model how likely the world is to be in state z_l given that report x_j is received from module M_i. When modeling functional modules is it however more natural to create the inverse relation, $P_i(x_j|z_l)$ so this will be the actual ingredient, or input, to the framework. Bayes Rule is then applied to create $P_i(z_l|x_j)$. It is important to notice that $P_i(z_l|x_j)$ in general is a function of various state variables, i.e., the performance of a module and thus its model will in general depend on "external" variables such as distances to objects, positional uncertainties etc.

A priori probabilities, $P(x_j)$ **and** $P(z_l)$ The a priori probabilities express a priori information about the reports and the state of the world, respectively. One can be found from the other, however, using $P_i(x_j|z_l)$, marginalization, and Bayes Rule. Thus, we will only specify a priori knowledge about the world, $P(z_l)$.

Costs, $C(A_i)$ This is the cost, measured in utils, of performing sensory action A_i.

This completes the description of the framework and the constituents for using BDA for sensor planning. However, there is still many gaps between this framework and the actualities of the "real world." In the next chapter those gaps will be described and directions for remedies will be pointed out.

Chapter 4

The Relation of BDA to a Navigation System

In this chapter various aspects of using Bayesian Decision Analysis for sensor planning in a real–world navigation system are discussed. The purpose of this discussion is to provide the context of the sensor planning in a more concrete manner but also to point out where there are discrepancies between the theoretical framework and a practical system.

It is evident that this discussion can not be completely general since navigation systems are implemented in various ways but it is believed that most of the issues discussed apply to a wide class of systems and in any case this illustrates the sort of engineering it often takes to fit theory and practice together.

4.1 Overall Control of Task Execution

The overall control of task execution provides the logical context for the sensor planner. This has indirectly been reflected in chapter 3 where terms as "task completion" have been widely used without a closer definition.

It should however be clear that when a task, as seen from the sensor planner, has been completed it does not mean that the robot has fulfilled all its purposes and can now be retired. Rather, some intermediate goal has been reached and the planner is now ready for a new task. This new task is determined by the task execution control (TEC) that again has a higher level (often user specified, thus called *user level*) task to execute. For the purpose of this discussion it is however adequate to consider the TEC the top of the hierarchy.

Here the TEC will be defined as a Discrete Event System (DES). The receipt of a user level task causes the TEC to go into a corresponding *task state* where it stays until the task is terminated by either a successful or un–successful completion. Then the DES returns to the idle state where it waits for user commands. This is illustrated in figure 4.1.

Each task state is in principle another DES that sequences the actual task execution. This again could be viewed as another set of DES'es etcetera—in other words, it is a matter of selecting

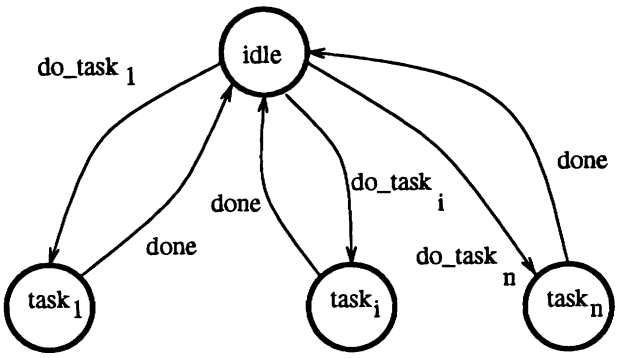


Figure 4.1: *The overall task execution control. This is a Discrete Event System that receives commands from the user level and executes them.*

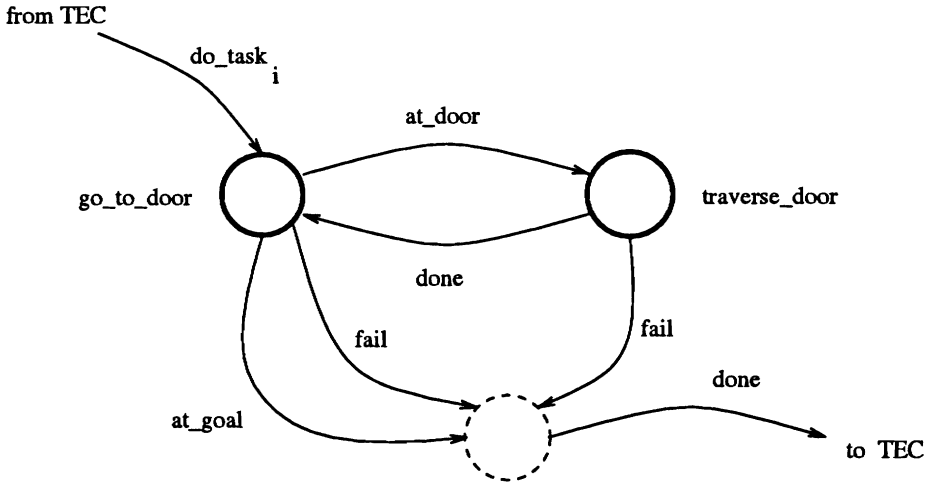


Figure 4.2: *The* drive–from–room–A–to–room–B task *modeled as a Discrete Event System with two states. The dashed "state" is not really a state but rather a virtual exit point introduced to make this DES look like state$_i$ in figure 4.1.*

a proper level of granularity of the description. Here we will generally assume that the task level as seen from the sensor planner (the *planning task level*) corresponds to a state in a task state DES in the TEC. Thus, if *task$_i$* is the *drive–from–room–A–to–room–B task* then it could be modeled as a DES with two states, *go–to–door* and *traverse–door*, as illustrated in figure 4.2. Each of those two states would then correspond to a task on the sensor planning level.

The determination the the planning task level is a compromise between the complexity of the task state DES'es in the TEC and the complexity of the sensor planning problem, i.e., the size of the decision trees. By selecting the planning task level very low the decision trees become small and thus the statistical modeling of the functional modules becomes less critical. This however means that the task state DES'es become increasingly difficult to construct since eventualities now not coped with in the sensor planning must explicitly be dealt with in the DES. Thus, the planning task level should be chosen so as to facilitate simple task state DES'es while at the same time keeping the statistical modeling and the computational demands of the sensor planning at a manageable level.

It is important to note, though, that even though there is a compromise between task state DES

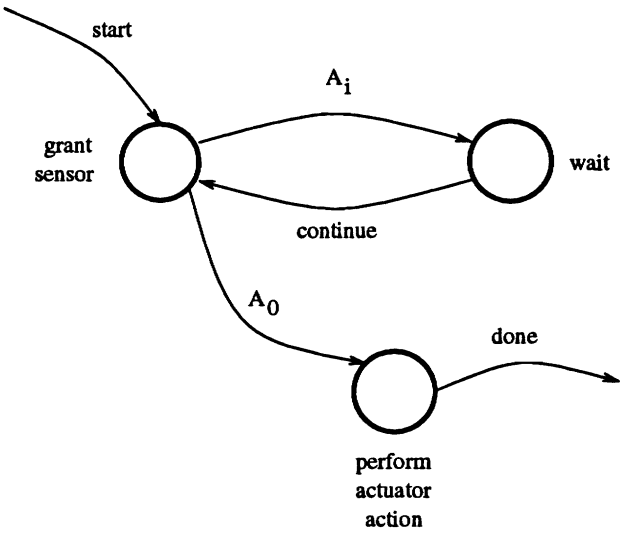


Figure 4.3: *The control structure for a planning level task, illustrated as a finite state machine. A_i means all actions except from A_0. The wait state can be implemented in various ways. This will discussed further in chapter 5.*

complexity and BDA complexity this does not mean that the system characteristics are invariant to tradeoffs in the description levels. This is due to the fact that DES'es (or rather finite state automata) are "binary" in the sense that you are either in a state or not. This relates somehow to traditional predicate logic where something is either true or not, and certainly, predicate logic can be used to build finite state automata. Bayesian Decision Theory, on the other hand, can as we have seen be used to evaluate alternatives against each other in a probabilistic manner which provides more flexibility and less of the traditional knowledge engineering than the DES approach.

What actually happens when a planning level task has been completed is that all sensors are *re–possessed* and control is resumed to the task state. Re–possessing the sensors from a purposive module largely corresponds to pre–empting the process, since the module stripped of its sensors can do nothing but try to get them back again. If the task state DES switches to a new task then the according decision tree is loaded into the sensor planner which then resumes control over the task execution by granting sensors to various modules. This scheme means that the system control is discontinuous at planning task changes. But it is at the very heart of Discrete Event Systems that the system abruptly changes character at discrete time intervals. However, by re–possessing the sensors before resuming control to the task state, it is ensured that no functional modules are active and thus the transition between tasks should be safe although not continuous.

The control associated with the sensor planner problem can be illustrated as the finite state machine shown in figure 4.3.

The simple idea behind the control strategy is that the vehicle should keep on collecting data as long as it pays and then eventually terminate the task by performing the actuator action. The control of the underlying purposive modules is then indirectly performed by granting and denying them sensors.

When the task is started (and before the planner enters the *grant sensor* state) the modules considered relevant for solving the task are informed that their services are now wanted. This means that these modules now can start requesting the sensors. The modules are however not obliged to do so, which means that the modules have some autonomy and can run asynchronously of the planner. Also if some modules break down or cannot get in contact with the planner this does not halt the system. In general, the requests for sensors are collected during the *wait* state and then evaluated in the *grant sensor* state. The timing of this is further discussed in section 5.2. When the task is completed, the sensors are, as previously mentioned, re–possessed but the relevant modules are also informed to stop requesting the sensors so that the planner is not burdened with bogus requests.

The technique used by the sensor planner to actually manage the sensors, i.e., to grant them and to re–possess them is described in section 4.2.

4.1.1 How a Decision Tree Maps to a Task State

It is not evident how a Bayesian decision problem maps to a task state and thus we will here give an example of such a mapping.

Let us consider the *traverse door* state in figure 4.2. When this state is entered, the sensor planner takes over the control. Note that the system is "at rest" when this happens since all tokens are held by the sensor planner.

In this somewhat hypothetical example, let us assume that there are 3 purposive modules, all using sonars, which can contribute to the door traversal. Also, let us assume that the sonars are mounted on turrets and thus are non–sharable. The purposive modules work are as follows:

Door Finder The door finder uses the sonars to locate the door frame which corresponds to localizing the robot's world position.

Door Scanner The door scanner uses the sonars in an active sensing strategy to accurately estimate the state of the door, i.e., whether it is open or closed. The active sensing strategy means that it is quite expensive to use the door scanner but the results are accordingly accurate.

Door Pinger The door pinger uses a very simple strategy to estimate the door state. This means that the information is not complete but quite cheap.

This constellation gives rise to a decision tree as illustrated in figure 4.4.

The decision tree basically illustrates that the sensor planning has the choice between 4 actions: to estimate the robot position, to buy the expensive but certain door state information, to buy the cheap but uncertain information, or to do none of that. Only in the latter case should the actuator action (i.e., driving through the door) actually be performed. The reports from the door scanner and the door pinger are different from the reports from the door finder. This deviates from the standard myopic structure as illustrated in figure 3.1. This will discussed further in section 4.5 but basically there is nothing alarming about this fact—it all fits well within the framework.

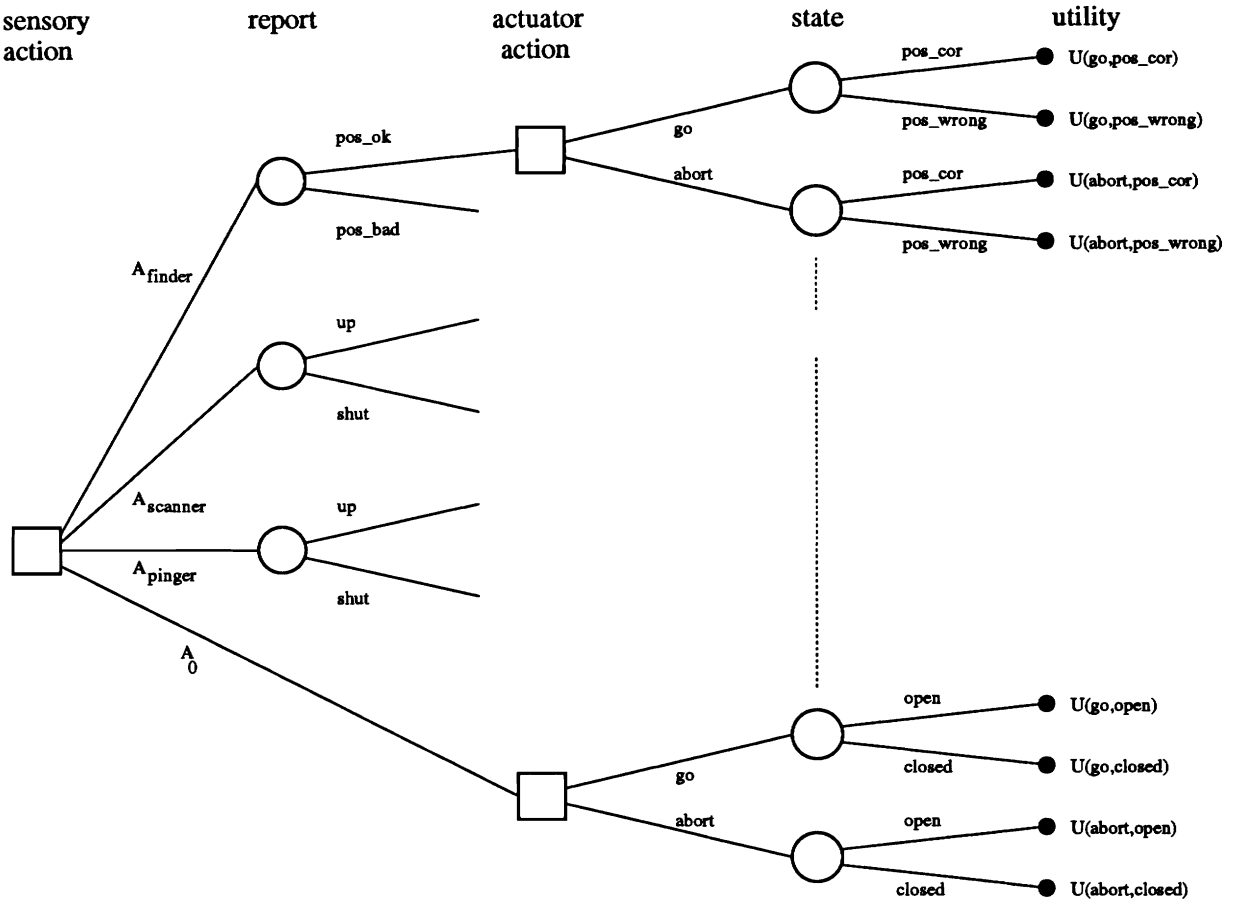


Figure 4.4: *The decision tree for the traverse door task. See text for further explanation.*

4.2 Control of Sensor Utilization

When the sensor planner grants a sensor to a functional module it must have some means of telling the module and of doing some book–keeping so that no other module is granted the same sensor at the same time–interval.

Operating systems theory provides a number of techniques for this purpose, most well–known is probably the use of semaphores. Here we will however adopt the notion of tokens and token passing since this is an intuitively compelling framework. Furthermore, this enables the use of the petri net framework in case it is found necessary to make a formal proof of the sensor utilization system.

The idea behind the token–passing based control is that when the sensor planner grants a module control over sensor S_i then the corresponding token, tok_i, is passed to the module that keeps it until it is done using the sensor or until the token is re–possessed from the module. It is here important to notice that the fact the tokens can be re–possessed means that the purposive modules should be designed to be preemptable.

A feature that makes the token framework attractive is that we can have sensors where the planner has several tokens for a given sensor. This corresponds to the situation where more

purposive modules can use the sensor at overlapping time intervals. This will typically be the case for, e.g., fixed sonars and other non–configurable sensors. There are several important reasons for not just omitting these sensors from the sensor planning but rather to account for them in a manner consistent with the configurable sensors. These reasons are:

Communication Although a sensor is in principle sharable between an unlimited number of agents, communication bottlenecks normally mean that a quite limited number of modules should be allowed to use it to ensure real–time performance.

System control The whole concept of controlling the system by granting and denying sensors to purposive modules means that the planner should control *all* sensors to ensure a coherent and safe behaviour of the system.

Costs Since there is a cost associated with using even an un–configurable sensor the use of the sensor should be evaluated by the sensor planning exactly like the use of configurable sensors.

If the sensor planner has N tokens for sensor S_j then this means that the N requesting modules with the highest expected utilities should each be granted a sensor given that $EU(A_i) > EU(A_0), i \in [1, m]$. This corresponds to modifying Bayes Decision Rule (equation 2.4) to selecting a set of N actions, given that EU_{BDR} for each of those N actions is larger than $EU(A_0)$.

In the previous sections the costs have been treated strictly according to the classical "static" framework. In a real world system there are however practical considerations about costs that should be discussed. This will be done in the following section.

4.3 Costs and Investments

In the real world where time is an important aspect, the notion of costs and utility should take on a slightly different meaning. This is due to the fact that it normally matters how long time it takes before a prize can be collected, especially since the costs normally must be paid in advance. This concern is basically what is behind the concept of an *investment.*

To give an example, is it better to invest 1 util to get an expected utility of 20 utils in 1 second or is it better to invest 70 utils to get an expected utility of 100 utils in 70 seconds? According to Bayes Decision Rule (equation 2.4) the latter option is the better one, since there is no notion of time in the framework. Even if (as hinted in the example) the cost is equal to spent time in seconds the latter choice is still preferred in the classical Bayesian framework. This is however not reasonable since a sensor normally can be "re–invested" after ended use and thus it can be expected that the first option is really more lucrative. To remedy this problem we will alter the meaning of maximum expected utility to a sense that will rather mean "optimal investment" although we will still use the term "maximum expected utility." Thus we will now re–define expected utility as the ratio between the "old" expected utility, without the cost subtracted, and the cost, or:

$$EU(A_i) = \frac{\sum_{j=1}^{n} P(x_j) EU'(x_j)}{C(A_i)} = \frac{\sum_{j=1}^{n} P(x_j) \left[\max_a \sum_{l=1}^{q} P_i(z_l|x_j) U(a_k, z_l)\right]}{C(A_i)} \tag{4.1}$$

where $C(A_i)$ is the cost of performing sensory action A_i, and EU' is the "traditional" expected utility. It is clear that this definition makes best sense if $C(A_i)$ is proportional to the time spent on action A_i and consequently we will in the following assume this is the case. The definition in equation 4.1 however poses a problem with the utility of action A_0 since the associated cost is assumed to be 0. What is interesting when making an investment is however not the utility after an elapsed time but rather the *increase* in utility compared to the situation where no investment was made. Thus, the "old" $EU(A_0)$ should be subtracted from the old $EU(A_i)$ before dividing by $C(A_i)$. This yields the following expression:

$$\begin{aligned} EU(A_i) &= \frac{\sum_{j=1}^{n} P(x_j)EU'(x_j) - EU'(A_0)}{C(A_i)} \\ &= \frac{\sum_{j=1}^{n} P(x_j)\left[\max_a \sum_{l=1}^{q} P_i(z_l|x_j)U(a_k,z_l)\right] - \max_a \sum_{l=1}^{q} P(z_l)U(a_k,z_l)}{C(A_i)} \end{aligned} \quad (4.2)$$

The quantity in the numerator is also called the *expected value of sample information (EVSI)* and can be shown always to be non–negative [11]. What we need now is however a way to decide when to stop buying information and perform the optimal actuator action. A way to overcome this problem is to select a ratio, Δ, which is the minimum utility (or "interest rate") we will accept and *define* this as the new $EU(A_0)$. Thus $EU(A_0)$ is:

$$EU(A_0) \equiv \Delta \ , \ \Delta > 0 \quad (4.3)$$

Note that with these definitions Bayes Decision Rule remains unchanged. The price for using this trick was the introduction of the constant Δ which has to be determined. This is however re–gained by the fact that with the new notion of expected utility the utilities and the costs need not be expressed in the same currency since only the ratio between them is relevant (in other words, Δ need not be dimensionless). This is an important feature that simplifies the modeling of the system. It is furthermore important to note that with this new definition of expected utility, the utilities can still (as in the classical theory) be scaled affinely according to equation 2.6 without this affecting the ranking of the $EU(A_i)$'s. $EU(A_0) = \Delta$ will have to be scaled accordingly, though.

So far, the theoretical discussion has focused on the situation where there is a single sensor being requested by a number of purposive modules. We will now discuss how this extends to the case where there are several sensors being requested by several modules.

4.4 Multiple Sensors

In a realistic navigation system, several sensors will be used concurrently. The sensor planner should be able to account for this to be of any practical use, and certainly to fulfill the demand for "intelligent" operation.

The straightforward way to account for several sensors is simply to create a decision tree, and thus a separate decision problem, for each sensor modality. It is important to note that the decision problems for each sensor would all be identical *except* for the cost, $C(A_i)$, associated with granting a sensor to a module. This is due to the fact that sensing cost depends on

the sensor and the amount of time the sensor is requested for while utilities and conditional probabilities are independent of this. It may seem surprising that the conditional probabilities do not depend on the actual sensor, but this is due to the fact that it is the purposive modules that are modeled and not the sensors themselves. There is however an exception to this which is treated in section 4.4.3.

The fact that all the decision problems are identical except for $C(A_i)$ means that we can actually treat all requests, R_i, uniformly no matter what sensors are requested and thus we can collapse all the decision problems into one in the following way:

1. For all requests, R_i, calculate $EU(A_i) \times C(A_i)$, i.e, the numerator of equation 4.2. This corresponds to evaluating the i'th branch of the decision tree and subtracting the result from the 0'th branch.
2. For all requests, R_i, calculate $C(A_i)$ and divide the value from 1 to get $EU(A_i)$.
3. Rank the expected utilities and grant requests as constrained by the number of tokens and Δ.

It is seen that all requests are evaluated using the same decision tree and thus the same information except from the cost, $C(A_i)$, of granting request R_i. The simplicity of this approach makes it a compelling way to perform the sensor planning but the simplicity also means that the approach has some constraints. These constraints are discussed in the following.

4.4.1 General Couplings

There is a fundamental, general coupling between the decisions to be made. This is due to the fact that evidence collected by selecting one action, i.e., by granting one sensor to a module, is propagated to subsequent decisions when the corresponding chance nodes are updated. Thus, if for example a purposive module using sonars updates the probability of a door being open from 0.5 to 0.7 then this is automatically accounted for in the next iteration when the decision tree is used to analyse what module should have, e.g., the camera. The system is however *not* able to account for two purposive modules gathering the same type of information concurrently. Therefore, the system can possibly exhibit a redundant behaviour. This is an expense of using the simple, myopic structure.

4.4.2 More Sensors Required for One Task

The first situation where it is too simplified to assume that there are no constraints between the decisions occurs in the case where a purposive module needs more that one sensor to accomplish a task. For example, this could be a path planning module relying both on sonar and visual sensing. Thus, the module must request both sensors to be able to accomplish something.

If the respective requests are treated independently then each of the corresponding expected utilities will reflect that the utility of granting *one* sensor will lead to the utility payoff. This corresponds to the situation where the expected utility is accounted for as many times as there are sensors needed for the task which is not a sensible thing to do. Under normal circumstances

(see section 4.4.5 for an exception) the correct thing to do is to account for the total sensing cost when evaluating whether a single sensor should be granted since the total sensing cost eventually has to be spent before either sensor is of any use. This corresponds to treating the request for the two sensors as one request using as cost the total cost of using both sensors. Therefore we will allow modules to request a set of sensors in a single request and consequently choosing an action can correspond to granting a set of sensors.

This means that equation 4.2 should be rewritten as:

$$EU(A_i) = \frac{\sum_{j=1}^{n} P(x_j) \left[\max_a \sum_{l=1}^{q} P_i(z_l|x_j)U(a_k, z_l)\right] - \max_a \sum_{l=1}^{q} P(z_l)U(a_k, z_l)}{\sum_{h\in\Omega} C_h(A_i)} \quad (4.4)$$

where $C_h(A_i)$ is the cost of granting sensor S_h to module M_i and where Ω is the set of sensors requested by module M_i.

If a module is granted a (proper) subset of the set of requested sensors, Ω, and it can not pursue its goal with this subset, then it should immediately return all tokens to the planner so that other modules can use them. An alternative method which is more efficient and which we will therefore use, is to assume that a module needs all the requested sensors in a set to function, and therefore the planner will either grant the whole set of sensors (if possible) or none of the sensors at all. This ensures that no sensors are temporarily wasted in partially granted sets.

4.4.3 One of More Sensors Required for a Task

If a purposive module, M_i, can solve a problem with one of a set of N sensors, e.g., with either a line striper or a stereo camera head, then the simple evaluation strategy can not be used, since the model $P_i(z_l|x_j)$ in general will depend on which sensor is used by the module.

We will circumvent this problem by simply modeling the purposive module for each of the N sensors and then present the module to the BDA as N distinct modules with fixed models instead of one module with one model conditioned on the sensor. The cost of doing this is that the same module can potentially get more than one of the sensors granted but this only means that it should return all but its favourite sensor.

Note that the effort of modeling the module is not increased by this strategy since the modeling has to be done for each sensor anyway.

4.4.4 One or More Sensors Useful for a Task

In some cases a module can solve a task using only one sensor but can get improved performance by fusing data from a set of sensor modalities. If we denote this set Ω and for simplicity assume that all combinations of sensors will be useful to some degree, then we could in principle solve the problem by modeling the module for each element in the power set over Ω, $\Omega^2 \setminus \emptyset$. This solution however differs from the solution in the previous section in that the module here now will compete *with itself* about sensor resources, which is clearly not optimal.

A way to solve this problem is to demand that the purposive module only requests sets of sensors that are mutually exclusive. For example the module can request $\{(linestriper \wedge camerahead)\}$

or $\{(linestriper) \vee (camerahead)\}$ but not both. This is however an "engineering solution" which is not entirely satisfactory since it would be preferable to incorporate the solution into the BDA framework in a coherent manner. How to do this is however not clear to me right now, and it should be looked further into.

4.4.5 Un–Employed Sensors

Un–employment is a general problem in the society, and it is no less of a problem in sensor planning when there are more sensors.

This is due to the fact that although it shows from a Bayesian decision analysis that the cost of using a sensor is not justified by the expected utility it can gain, then it may actually be better to use the sensor for something than to let it be idle. This would be true in a situation where the task is not completed and thus other sensors are still employed. Then the use of time as a measure of cost is not really valid since the time will be spent anyway. There may still be a cost due to power consumption and computational cost but often the most important component in the calculation of $C(A_i)$ is time. Similarly, if a module, M_i, requests a set of sensors, $\{S_1 \wedge S_2\}$, and sensor S_2 would be idle if not used by M_i then a more reasonable measure for the cost of action A_i would be $\max\{C_1(A_i), C_2(A_i)\}$ rather than $C_1(A_i) + C_2(A_i)$ as indicated by equation 4.4.

What to do about this is not really clear by now, but maybe an explicit notion of time can alleviate the problem.

4.5 The Myopic Decision Theory

The effects on the theoretical framework of choosing a myopic decision theory were largely discussed in chapter 3. In a practical context there is however an important problem with this policy that must be addressed.

The problem is a direct consequence of the fact that the myopic planning only looks one step ahead. In section 3.3 a verbal formulation of the myopic strategy was given as:

> "given that at most one sensory action can be performed before task completion, which one should this be in order to maximize the expected utility?"

The problem with this is that sometimes there are necessary actions to perform that do not directly increase the expected utility, but which are necessary as support for those modules which do. We will call modules performing such actions *support modules* (and the other modules *productive modules*). A typical example of a support module is one performing landmark recognition. If the vehicle does not know where it is then only a few other modules can do something purposeful. But the process of performing landmark recognition does not directly increase the knowledge of, e.g., the state of a door and thus it does not directly increase the expected utility of traversing the door. But without the landmark recognition those modules which *can* estimate the state of the door cannot operate since they would not know where to look.

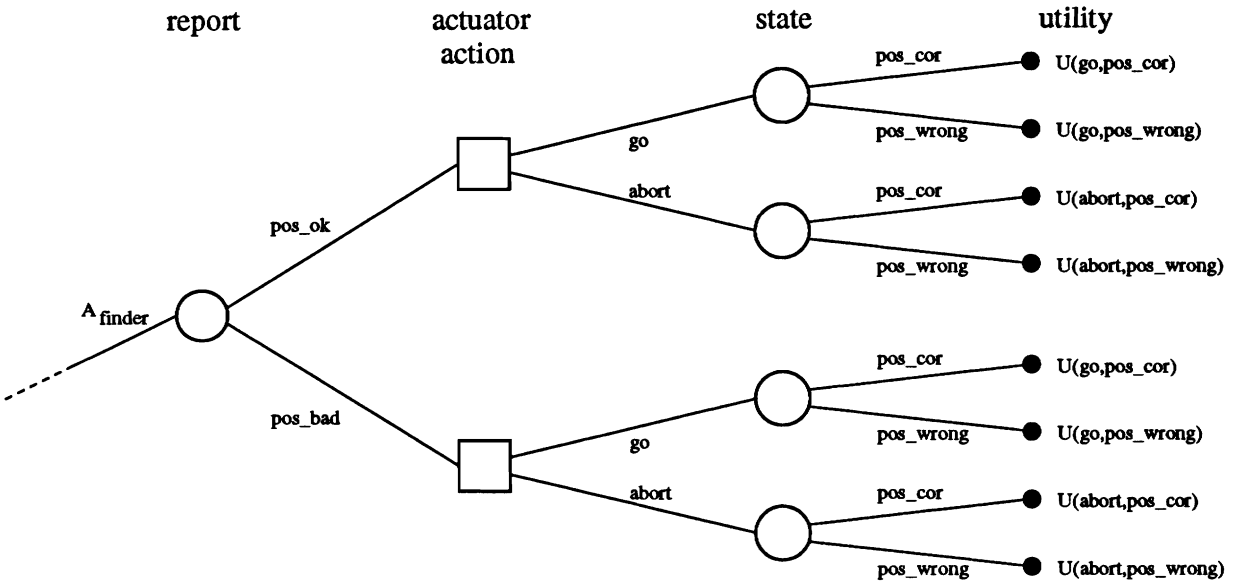


Figure 4.5: *An example of a part of a decision tree showing the context of a support task.*

Thus, there is a need for a mechanism to give support modules proper credit without resorting to planning several steps ahead.

In the example given in section 4.1.1 a solution to this problem was indicated. The trick is to introduce additional context in the form of sets of world states and utilities that can "lure" the system into granting the support modules sensory input. The reason for this being possible is due to both the control strategy of the sensor planning and, more indirectly, to the fact that we have a known expected utility of A_0, $EU(A_0) = \Delta$ (see section 4.3).

What we want the system to do is to work on the "real problem" when that is possible and to use the support modules only when necessary. This is somewhat similar to a *depth first* search strategy. For a given support module, M_{sup}, with an associated model, $P_{sup}(X|Z)$, we can *define* or *engineer* the associated utilities, $U_{sup}(a, Z)$, to make the support module "win" the sensors when it can contribute with information and when the productive modules can not.

To illustrate a way to do this is best done using an example. In this case let us re–visit the example from figure 4.4. This is shown in figure 4.5.

First of all, we must decide upon what we want. In this case, we would like the door finder to be employed whenever the positional uncertainty is too large for any of the productive modules to work satisfactorily. We can thus define a maximum uncertainty, or error, e_{max}, which we can tolerate. This e_{max} can be found as $\min(e_{max,pinger}, e_{max,scanner}, e_{max,traversal})$ where $e_{max,traversal}$ is the maximum tolerance on the vehicle position when actually traversing the door. When the door finder succeeds in reducing the error to less than e_{max} the report will be *pos ok* and when not it will be *pos bad*. We would thus like $EU(A_{finder})$ to be greater than Δ when the positional uncertainty is larger than e_{max} and smaller than Δ when the uncertainty is smaller than e_{max}. The latter is important since we do not want to run the door finder when other modules can run, and if $EU(A_{finder})$ is never smaller than Δ the actuator action will never be performed. The desired value for $EU(A_{finder})$ as a function of the uncertainty, $|uncert|$ is illustrated in figure 4.6. The reason for the upper "cut off" is that when the uncertainty becomes large, the

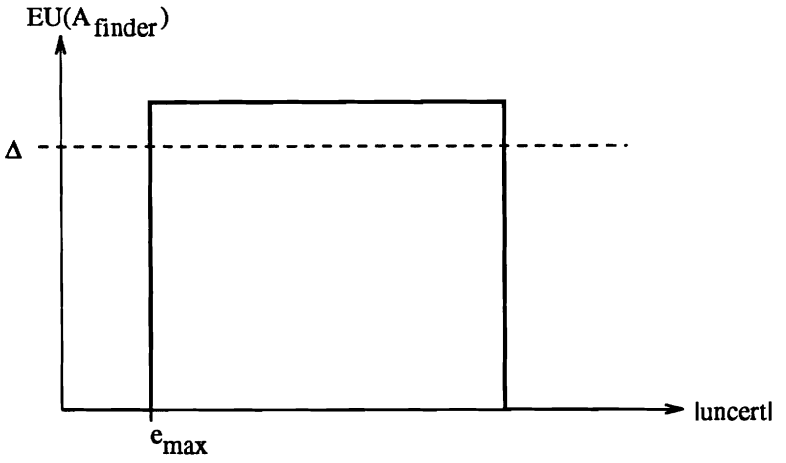


Figure 4.6: *The desired expected utility of the door finder module as a function of the positional uncertainty of the robot.*

door finder can not find the door and thus it should not be run.

A quick way to solve the problem would then be to simply define $EU(A_{finder})$ according to figure 4.6. This is however undesirable for two reasons: first we would like the calculation of $EU(A_{finder})$ to be consistent with the BDA framework. Second, by engineering the utilities we obtain that $EU(A_{finder})$ is dependent of the model $P_{finder}(X|Z)$ which means that if this model changes (due to learning or other reasons) then the expression for $EU(A_{finder})$ is automatically adjusted.

There are two ways to determine the right utilities. One is to determine them by experimentation, the other to use the models, $P(X|Z)$, and numerical or graphical methods to adjust the utilities until the shape of $EU(A_{finder})$ is (qualitatively) similar to that depicted in figure 4.6.

This is clearly a tedious method, but the advantage of this method is that (apart from solving our problem) everything is formulated in the BDA framework and that the a priori probabilities and the models can change at least quantitatively without rendering the utilities invalid. Another benefit is that the same scheme can be used for all tasks where the door finder is used as a localization module. This is convenient since while the productive modules can normally only be used for a single task so can the support modules often be used for a variety of tasks.

Chapter 5

Timing Issues

In a real world robotics system the notion of time is un–avoidable. Things have to operate in real time to ensure useful and safe operation. The problem with Bayesian Decision Theory is that it does not inherently contain a notion of time. So far, time has only entered the framework indirectly by considerations of costs. There are however a variety of other places where time should be considered explicitly in the sensor planner.

Although the notion of time in most instances can be considered as something "implementational" it is none the less important to discuss time explicitly to see what the possible implications for the decision framework may be. For example, in section 4.1.1 it was described how a task was completed when it was found that the optimal action was A_0. This is a time–less definition which does not account for such eventualities as the fact that maybe A_0 was optimal because no modules had time to submit requests or maybe A_0 was only *temporarily* the best action. On the other hand, some action A_i, $i \neq 0$ may turn out to be the best action each time for a longer period, which is an indication of some error or maybe cyclic behaviour which should be detected and the task stopped. This and similar subjects will be discussed in the following.

5.1 An Explicit Notion of Time?

The first question that comes to mind is whether the whole system should be "clocked" and have a common, explicit notion of time.

We think this should be avoided for several reasons. First of all would this kind of operation severely limit the autonomy of the various purposive modules. Second we would like the system to be distributable over a number of different computers which need not all be true real time systems. How to implement a common clock in such systems is still a research issue and certainly not something we would like to take on in this project. Therefore the system should be designed to be flexible to time variations and only operate with time intervals (or relative increments).

5.2 Timing of Sensor Requesting

To make the system more tolerant to module failures and to make the various purposive modules more autonomous it has been chosen to let the modules request the sensors at will rather than letting the sensor planner solicit requests and wait for answers. This asynchronous type of operation however calls for a method to determine when to stop waiting for requests and start planning. The transition from receiving requests to planning is very important since the fundamental idea behind the system is that the modules *compete* for the resources, and thus if the planner starts planning before all relevant modules have made their "bids" then there is a severe risk that the system will not function optimally. Therefore care has to be taken that all relevant modules have a chance to submit their requests before the planning starts while the planner at the other hand cannot wait infinitely for requests since this would stall the system.

It can be expected, though, that the modules that want the sensors and have not got them will request them as soon as possible after they have received the latest message from the planner, which could either be a notification that their services are now wanted or a reject of a previous request. Thus, what the planner has to compensate for are delays due to internal processing of the previous message in the modules and communication delays.

To do this, a strategy which we call the "bus driver's strategy" has been selected. Basically what this strategy does is to wait (in principle infinitely) for the first request and then keep waiting as long as new requests keep arriving. The analogy with a tour bus driver is that she would wait for the tourists to come back to the bus, but when the stream of tourists dies out and no tourists have arrived for a time interval she will assume that all tourists interested in continuing have boarded the bus and thus she will take off. The reason why this strategy has been selected is that the requests are believed to arrive quite simultaneously except for differences in processing and communication delays just like a group of tourists that know when the bus leaves but can get delayed due to a queue at the restroom. The strategy is illustrated in figure 5.1.

It is clear that in reality it is not feasible to let the algorithm wait infinitely for the first request since this would bring the system to a deadlock if the purposive modules either were broken or did not want to request any sensors. Thus, the first wait should also time out after a period, T, and make the planner enter the planning phase. When no requests are received this would provoke a termination of the task by performing the currently optimal actuator action.

The problem with the "bus driver strategy" is to determine an appropriate length, Δt, of the time interval to wait before start planning. The longer the interval the larger probability that all requests have arrived, but a longer interval also increases the response time of the planner thus slowing down the entire system.

A scientifically correct method would be to model the arrival of requests statistically as, e.g., a Poisson process and then calculate the interval, Δt, that would assure that only $\alpha\%$ of the requests would be missed. This however just shifts the problem from determining Δt to determining α which is not trivial since what is the loss associated with missing a request as opposed to slowing down the planner? As a consequence it has been chosen simply to determine Δt by experimentation (although I prefer to call it "machine learning of a variable using human feedback").

It is clear that other strategies for the timing of sensor requests could have been chosen. Whether

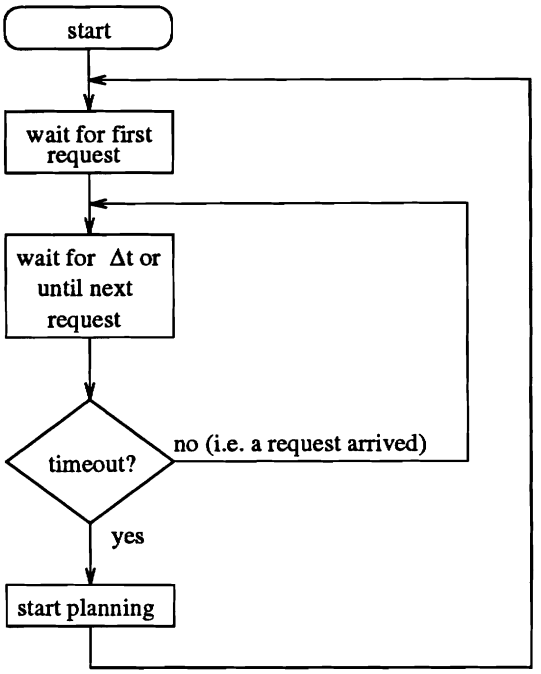


Figure 5.1: *Flow diagram for the "bus driver strategy." Δt is the time the "bus" waits after the latest "passenger" has arrived before it continues.*

other strategies will be more adequate will however be difficult to establish without practical experience, and probably the "optimal" strategy will depend on the actual system architecture, hardware etc.

5.3 Progress Watch/Task Completion

An important concern regarding most robotic systems is that we want to be sure that the tasks are completed in finite time. This basically means that deadlocks and cyclic behaviour should be avoided. These two problems are however very different in nature.

A deadlock is basically a local feature where a single component stalls the entire system. Thus possible deadlocks should be envisioned and avoided locally. Cyclic behaviour on the contrary is a global feature where the entire system repeats the same behaviour over and over again. This means that cyclic behaviour is much harder to detect and avoid since it takes some central progress watch mechanism to detect and break it. The only central component in a system of the type we consider here is the sensor planner. Therefore it is here that we should detect and break cyclic behaviours. This seems quite natural because one can argue that since the sensor planner basically is the control of the task execution it is also the one that is responsible for *creating* possible cyclic behaviours.

The most simple form of cyclic behaviour would be if the same purposive module got the sensor resources each time for an unbounded number of times. This can be avoided in two ways, either the planner can have a maximum number of times per task a module can get a request granted or it can be demanded that when a purposive module is run, the result of this should always

alter the state variables on which the model, $P_i(z_l|x_j)$, depends. This means that eventually the conditions for running a module would have changed sufficiently for it not to be run anymore.

While the latter approach is more appealing since it fits more naturally into the BDA framework it cannot guarantee that cyclic behaviors will not occur. Thus a combination of the two methods should be used where the limit of grants per task can be considered as a safety device.

5.4 Preempting of Modules

In section 4.2 it was stated that the purposive modules should be preemptable in order for the planner to re–possess the tokens at the completion of a task. When we consider the fact that the sensor granting happens as a sequence over the time span of the task execution there may however be other reasons to preempt a process. This is due to the fact that the state variables on which a given module model depends may change *during* the execution of the module, especially as a result of other modules gathering evidence.

Therefore, the basis on which a module was granted its sensors may be rendered invalid *while the module is still using the sensors.* Thus it may be relevant to re–possess the sensors, i.e., preempt the module before it has completed its actions. This indicates that the decision of granting a module a set of sensors may not just be a decision made at a point in time but rather that the decision should be maintained as long as the module still has the sensors. Thus, each time the sensor planner evaluates the received requests it should also re–evaluate the previous requests it has granted which are not yet expired, i.e., where the sensors have not yet been returned. This means that the planner should keep a record of not only who got the sensors but also how and when.

It must however be the case that old requests should be re–evaluated using as a measure of cost the time *left* of the requested time rather than the originally requested time. This is due to the fact that the expected utility of granting the sensors is now only $t_0 - \Delta t$ time units away from being awarded, where Δt is the time elapsed since the grant of the request and t_0 is the originally requested time. With this entered into equation 4.2 or equation 4.4 it can be seen that the expected utility of retaining the decision of action A_i goes towards infinity as Δt goes towards t_0 (in the case where $C(A_i) = t$). This is intuitively appealing since we do normally not want to preempt a module just before it finishes processing. Maybe the idea of $t_0 = \Delta t$ as a singularity is less appealing but that can be taken care of by not re–evaluating a request if $t_0 - \Delta t < \epsilon,\ \epsilon > 0$.

In the discussion above it has been assumed that the algorithms used by the purposive modules are so–called *contract* algorithms [14] where the algorithm has a fixed, allocated amount of time (t_0) at its disposal. There is however a class of algorithms called interruptible anytime algorithms that can be preempted at a given time and still produce meaningful results although the quality of these results improves as a function of time. This is a very interesting class of algorithms that we however not will discuss further in this work.

The here described preempting scheme can be regarded as the Bayesian Decision Theory version of non–monotonic reasoning and is as such quite pleasing since it fits naturally into the whole framework, given that there exists a notion of time (and apparently this is more than can normally be said of traditional non–monotonic reasoning).

Chapter 6

Description Language

To facilitate easy and modular design of the sensor planning, a description language, or formalism, has been made to describe the sensors, the tasks, and the purposive modules, respectively. In general, the description language has been made with the intention of creating a high level description language that at the same time supports low level control and a computationally efficient implementation.

The description language and the philosophy behind it is described in the following sections, and examples of actual description files are given in appendix A.

6.1 Sensor Description

The description of the available sensors is, to some surprise maybe, by far the simplest part of the language. Traditionally, great effort has been put into describing sensors in order for the various systems to be able to decide the appropriateness of applying a sensor to some problem. However, as previously mentioned, this kind of description is not relevant for the sensor planner described here, since what matters is how suited the purposive modules are for a task and not how suited a sensor is for a purposive module (although the latter of course influences the former). Thus, all the sensor planner needs to know about the sensor configuration is what sensors are present, their names, how many processes can share them, and what it costs to use a sensor for a given time interval.

Therefore, if Φ denotes the set of available sensors, then each element, $\mathcal{S}$, in Φ is a 3–tuple $\{\phi, N, C(t)\}$ where ϕ is the name of the sensor, N is the number of tokens for $\mathcal{S}$, and $C(t)$ is a function expressing the cost of using $\mathcal{S}$ for t time units. Thus if we have S sensors in the system, the description would look like:

$$\Phi = \{\{\phi_1, N_1, C_1(t)\}, \ldots, \{\phi_i, N_i, C_i(t)\}, \ldots, \{\phi_S, N_S, C_S(t)\}\}$$

The reason for associating a name with each sensor is that they can then be referenced by name and not by their index, i, which means that the modules requesting the sensors do not need to have knowledge about the index of the sensor in the description, but only their names which is

more invariant and intuitively pleasing. It is for the same reasons that also task and module descriptions have names associated.

6.2 Task Description

The task description defines the various planning level tasks (see section 4.1). For each task, $\mathcal{T}_i$, in the set of tasks, Λ, the description is a 6–tuple:

$$\mathcal{T}_i = \{\lambda_i, \Pi_i, \mathcal{A}_i, \mathcal{Z}_i, \Delta_i, \mathcal{U}_i\}$$

where the elements have the following meanings:

λ_i is the associated name of task $\mathcal{T}_i$.

Π_i is the set of modules that can participate in the fulfilling of the task. This means that this is the set of modules from which sensor requests will be accepted and each element is the name of a purposive module. Note that this is *not* the same as the name of a module model, since the same purposive module can be evaluated according to different module models, depending on what sensors it requests (see section 4.4.3).

$\mathcal{A}_i$ is the set of actuator actions that can complete the task.

$\mathcal{Z}_i$ is the set of world states that influence the utility of completing task $\mathcal{T}_i$.

$\Delta_i \equiv EU_i(A_0)$ is the minimum accepted "interest rate". Δ_i can be any *function* but normally this would be a constant.

$\mathcal{U}_i$ is the set of utilities, $U_i(z, a)$, defined $\forall a \in \mathcal{A}_i, \forall z \in \mathcal{Z}_i$. This is in general a set of functions, which enables the utilities to depend on state information and other evidence.

6.3 Module Description

The module description basically holds the probabilistic models of the purposive modules. This is all the sensor planner needs to know about the modules in order to make its decisions. The set of module models is called Ψ and each element, $\mathcal{M}_j$, is a 4–tuple,

$$\mathcal{M}_j = \{\psi_j, \mathcal{X}_j, \mathcal{Z}_j, \mathcal{P}_j\}$$

where the elements have the following meanings:

ψ_j is the associated name of module model $\mathcal{M}_j$.

$\mathcal{X}_j$ is the set of reports possibly generated by the module.

$\mathcal{Z}_j$ is the world states that the reports $\mathcal{X}_j$ depend upon, or "says something about."

$\mathcal{P}_j$ is the set of conditional probabilities, $P_j(x|z)$, that makes up the probabilistic model of the module. $P_j(x|z)$ is defined $\forall x \in \mathcal{X}_j, \forall z \in \mathcal{Z}_j$. Since $P_j(x|z)$ in general depends on state information this is a set of functions.

For a purposive module ($\in \Pi$) to be able to request a set of sensors, $\Omega \subseteq \Phi$, for solving task $\mathcal{T}_i$ it must hold for the according module model, $\mathcal{M}_j$, that $\mathcal{Z}_j \subseteq \mathcal{Z}_i$. In other words, the reports, $\mathcal{X}_j$, generated by the module must be related (by $\mathcal{P}_j$) to the world states of relevance to the task.

When a purposive module wants to request a sensor all it has to specify in the request is the set of sensors, $\Omega \subseteq \Phi$, how long time each sensor will be used, $\overline{t}$ (where $\text{Rank}(\Omega) = \text{Rank}(\overline{t})$), and the module model, $\mathcal{M}_j$. Thus, a sensor request, R, is a 3–tuple, $\{\Omega, \overline{t}, \mathcal{M}_j\}$. From this information it is then possible for the sensor planner to calculate the expected utilities and therefore decide which modules (if any) should have their requests granted.

6.4 Syntax of Description Files

A separate description file for each of the sensor, task, and module descriptions exists. The structure of the description files is however similar and is quite alike the syntax for `yacc` files. This style has been adapted because it allows for relatively abstract descriptions to be supplemented with C–code where necessary.

In general a description file has 3 sections: A preamble, a main body, and a functions section. The sections are separated by a **%%**–marker.

The preamble contains two things: C–code header information (enclosed in %{ and %}) necessary for compiling the functions part, and "declarations" of variables in the main body. These declarations have the format '`%id var var var....`'.

The main body contains the descriptions of either the sensors, the tasks, or the module models. This description is in general a high–level symbolic description which however can make references to variables and C–functions. These referenced functions are then declared in the last part of the file, the functions part.

Thus, the description files contain both abstract descriptions and old fashioned C–code. This duality may seem peculiar but it offers an opportunity to unite abstract descriptions with control and "ingenuity" that can only be obtained at a lower specification level.

In the following, text in square brackets ([text]) denotes optional contents. Words in bold face (**text**) are keywords.

6.4.1 Syntax of the Sensor Description File

The syntax of the sensor description file is as follows:

%{

[C-code header information]

%}

%%

sensor "ϕ_1" { N_1 "$C_1(t)$" }
⋮
sensor "ϕ_S" { N_S "$C_S(t)$" }

%%

[C--functions]

Note that there are no variable declarations in the sensor description file. The cost functions, $C_i(t)$, can either be written directly "in–line" or can be referenced by a function call to a function declared in the C–functions section. This is a standard way of describing functions that is also used in the other description files.

6.4.2 Syntax of the Task Description File

The syntax of the task description file is as follows:

%{

[C-code header information]

%}

%aa a_1 a_2 ... a_P
%ws z_1 z_2 ... z_Q

%%

task "λ_1" {
    modules { π_{11} π_{12} ... π_{1m} }
    actuator_actions { a_{11} a_{12} ... a_{1p} }
    world_states { z_{11} z_{12} ... z_{1q} }
    delta = "Δ_1"
    utilities {
        U(a_{11} , z_{11}) = "$u_{11,11}()$"
        U(a_{12} , z_{11}) = "$u_{12,11}()$"
        ⋮
        U(a_{1p} , z_{1q}) = "$u_{1p,1q}()$"
    }
}

⋮

task "λ_T" {
modules { π_{T1} π_{T2} ... π_{Tm} }
actuator_actions { a_{T1} a_{T2} ... a_{Tp} }
world_states { z_{T1} z_{T2} ... z_{Tq} }
delta = "Δ_T"
utilities {
U(a_{T1} , z_{T1}) = "$u_{T1,T1}()$"
U(a_{T2} , z_{T1}) = "$u_{T2,T1}()$"
⋮
U(a_{Tp} , z_{Tq}) = "$u_{Tp,Tq}()$"
}
}

%%

`[ C--functions ]`

The actuator actions declared with the **%aa** statement, $\mathcal{A}$, are the union of all actuator actions from all tasks, i.e., $\mathcal{A} = \cup_{i=0}^{T}\mathcal{A}_i$. Similarly, for the world states declared with the **%ws** statement $\mathcal{Z} = \cup_{i=0}^{T}\mathcal{Z}_i$. Δ_i is a function just as the utility functions, $u_{ij,ik}()$. It is important to note that the spaces in the syntax are also part of the syntax, i.e., they can not be omitted.

6.4.3 Syntax of the Module Description File

The syntax of the module description file is as follows:

%{

`[C-code header information]`

%}

%r x_1 x_2 ... x_N

%%

mmodel "ψ_1" {
reports { x_{11} x_{12} ... x_{1n} }
world_states { $z_{11'}$ $z_{12'}$ ... $z_{1q'}$ }
probs {
P(r_{11} , z_{11}) = "$p_{11,11}()$"
P(r_{12} , z_{11}) = "$p_{12,11}()$"
⋮
P(r_{1n} , $z_{1q'}$) = "$p_{1n,1q'}()$"
}
}

⋮

mmodel "ψ_M" {
reports { x_{M1} x_{M2} ... x_{Mn} }
world_states { $z_{M1'}$ $z_{M2'}$ ... $z_{Mq'}$ }
probs {
P(r_{M1} , z_{M1}) = "$p_{M1,M1}()$"
P(r_{M2} , z_{M1}) = "$p_{M2,M1}()$"
⋮
P(r_{Mn} , $z_{Mq'}$) = "$p_{Mn,Mq'}()$"
}
}

%%

`[ C--functions ]`

The reports declared with the **%r** statement, $\mathcal{X}$, are the union of all reports from all modules, i.e., $\mathcal{X} = \cup_{j=0}^{T} \mathcal{X}_j$. The world states that these reports tell something about, $\{z_{j1'}, \ldots, z_{jq'}\}$, are a subset of $\mathcal{Z}$.

Chapter 7

Experiments

To verify that the ideas presented in this work can actually be applied to real world autonomous robot navigation, a series of experiments were conducted as proof of concept.

The experimental platform, SensorBot, is a Labmate base equipped with the following sensor modalities: a structured light linestriper with an associated CCD camera, a stereo camera pair, a ring of 16 infra–red detectors, and a ring of 16 acoustic Polaroid sensors. In the experiments, only the sonars and the linestriper were used, though. SensorBot is shown in figure 7.1.

The scenario used for the experiments was that SensorBot should pass a doorway. However, due to the fact that the robot was too wide to pass the door, the task was reduced to locating the doorway and estimating whether the door was open enough for an imaginary, smaller robot to pass.

Three purposive modules were built to solve the task. These were:

Door finder This module uses an active sonar sensing strategy for locating a doorway when the robot is already close to the expected position of the doorway. Although the sonars are not configurable by themselves an active strategy can be obtained by moving the entire vehicle while sensing. The door finder utilizes a priori model knowledge to guide the sensing strategy. Altogether this means that the door finder is capable of locating the door with relatively high precision although the uncertainty on the robot start position is "large." Due to the active sensing strategy, it is however rather costly to use the door finder since the necessary movements of the robot take some time.

β estimator This module uses the linestriper to estimate the angle, β, of the door with respect to the doorway. Since the linestriper is fixed with respect to the vehicle, the vehicle has to move to provide a good viewpoint for the linestriper. Thus, this module also uses an active sensing strategy although not as elaborate and thus expensive as the door finder's. The β estimator uses a priori model information to guide the movements of the robot. Due to the relatively high resolution of the CCD camera the linestriper delivers quite accurate estimates of β but the position of the robot has to be known with fair accuracy for getting the doorway into the field of view of the linestriper. While moving, the β estimator uses the sonars to check the path.

Figure 7.1: *SensorBot. The thing on the top that resembles an overhead projector is the linestriper.*

module	cost	output	range
door finder	high (100 sec)	quite accurate	large
β estimator	medium (8 sec)	accurate	medium
door pinger	low ($\leq$ 1 sec)	incomplete	small

Table 7.1: *Most important features of the purposive modules. By "range" is meant the tolerable size of the uncertainty on the vehicle position.*

Door pinger This module uses a very simple, passive sonar sensing strategy for detecting if a door is closed. The module simply tries to fire a sonar through the door opening and if a return signal within certain bounds is received, the door is assumed to be closed. The door pinger can however not detect whether a door is open (in the sense that it is open enough for a robot to pass) so it can only deliver incomplete but very cheap information. Due to the poor angular resolution of sonars the position of the vehicle has to be known with high accuracy for making sure that the door pinger really pings towards the door opening and not, e.g., a door post.

The three purposive modules are explained more in technical detail in appendix B, C, and D so here we will only list the major features, see table 7.1.

A block diagram of the test system is shown in figure 7.2. For the experiments, the planner was given probabilistic models of the three purposive modules reflecting the characteristics listed in table 7.1. The cost, $C(A_i)$, of using a module was equal to the expected time consumption in

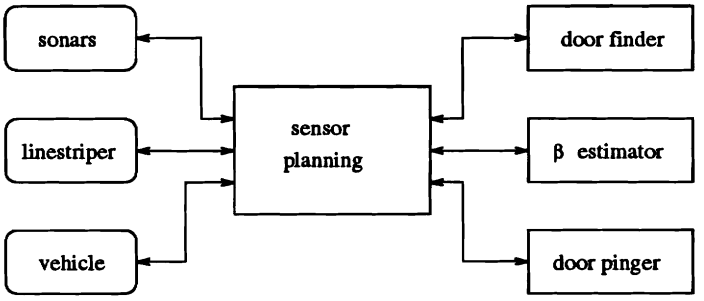


Figure 7.2: *Block diagram of system used in experiments.*

type	β [deg]	uncertainty	actions	actuator action
1	5	small	pinger	abort
2	45	small	pinger, β estimator	abort
3	90	small	pinger, β estimator	go
4	5	medium	β estimator	abort
5	45	medium	β estimator	abort
6	90	medium	β estimator, finder, pinger	go
7	5	large	finder, pinger	abort
8	45	large	finder, pinger, β estimator	abort
9	90	large	finder, pinger, β estimator	go

Table 7.2: *Results from experiments. β denotes the angle between the door and the door frame. The door was considered closed, i.e., the imaginary robot could not pass when $\beta < 60$ degrees.* The uncertainty on the vehicle position must be small for the robot to pass the door.

seconds. The models, $P(x_j|z_l)$, were derived empirically from experiments with the respective modules *in isolation* and were in general functions of the uncertainty on the robot position, thus reflecting the range of the module. For the experiments only piecewise linear models were used for the modelling. The utilities were also found experimentally. Normally, when BDA is applied to economics an analyst interviews the decision maker (e.g., a manager) in order to figure out her utility functions. In these experiments that interaction was performed between the robot and the designer in order to obtain the utilities. Thus the system "interviewed" the designer by showing examples of performance with different utilities which were then modified (by the designer) until the performance seemed in accordance with the goals of the designer. It is however important to notice that only one set of utilities must be determined for all the productive purposive modules. This can in principle be done by using only one purposive module. Apart from that some extra utilities must be determined for the support modules, in this case the door finder, though.

Since all the modules used the sonars and the sonars for the sake of the experiment were defined as being un–sharable ($N_{sonars} = 1$) the sonars were a critical resource and thus only one module could run at the time. In table 7.2 test results are listed. The table lists the state of the world, the selected actions and the final actuator action concluding each run.

Several experiments of each type were conducted and the system performed robustly the way listed in table 7.2. The only exception was with experiments with $\beta = 5$ degrees (type 1 and 7) where the pinger sometimes did not detect the door which caused the β estimator to get the sensors afterwards.

The general result from the experiments is that the sensor planner plans the sensing actions in an intuitively pleasing way where it tries the cheapest possible sensing first and stops when it is asserted that the door is either closed or open and the position is known well enough for the robot to pass. One remarkable exception from this is in the type 6 experiment where the pinger is employed at the end, although the β estimator has detected the door as open. This is due to the fact that a little extra certainty, and thus expected utility, can be gained for a very low price by employing the door pinger. This behaviour was not expected but "emerged" and seems sensible.

Because only 3 purposive modules were used, the example seems somewhat trivial in the sense that the planning could also have been performed with, e.g., a rule based planner. It is however interesting to notice that because no rules were used this approach can be expected to scale better since new sensors, tasks, and purposive modules can be added to the system without any modification of the existing system. Especially, it would be interesting to break with the tradition of only having one purposive module for each purpose and to create a set of modules using different cues and sensors to perform the same task. This situation can be expected to be well handled by the BDA framework since it automatically selects the modules currently best suited for a given task. This has partly been demonstrated with the β estimator and the door pinger that do the same thing in different ways.

Chapter 8

Conclusion

In this report, work on an approach for sensor planning using Bayesian Decision Theory was presented. The context in which the sensor planning problem was addressed was that of autonomous mobile robot navigation. The fundamental structure of the navigation system was assumed to be an architecture with purposive modules using active sensing strategies. The sensor planner described is a unified approach for treating both sensors and actuators. Also, the sensor planner was used to control the execution of the purposive modules in the sense that granting a purposive module sensors is the same as asking it to start processing and vice versa.

It was chosen to use the BDT framework to obtain a more responsive and modular system than is the case when using traditional planning techniques. Also, BDT was chosen for its inherent ability to reason about uncertainty.

The modularity of the system arises from the fact that sensor modalities, tasks, and modules can be described separately and that no rules (except from Bayes Decision Rule) are used to guide the behaviour.

The notion of time was entered into the framework as a necessary concept for making the sensor planning applicable to real world robot navigation. Furthermore, the cost of a sensory action was equaled with the time spent here–on. This is assumed to be reasonable since it is yet not known how to create "realistic" cost functions, but it can be expected that time (and linearly related terms) will be dominating in most such cost functions.

Experiments have shown that a sensor planner implemented on basis of the concepts presented in this report is capable of controlling the sensors (and thus the functionality) of a mobile robot performing a real world task.

So far, the test scenarios have been relatively simple since only a small number of purposive modules have been competing for the sensors. Future work will show how the method generalises to other, more complex scenarios where more modules are present. Especially, we would like to see how well the system copes with more modules doing "the same thing" but with different techniques using different cues and sensors. In other words, we would like to be able to manage "a bag of tricks" contrary to the traditional, rather monolithic systems with only one module for each purpose.

Chapter 9

Further Research Topics

As described in chapter 7 the experiments conducted can only be considered as a proof of concept. Therefore more work has to be done in order to see how the results generalise and scale. There are however also a number of other topics that could deserve some attention. These are briefly described in the following.

9.1 How to Get Probabilities, Models, and Utilities

It is evident that a serious problem associated with the BDA framework is that there seems to be no specific way to come up with the ingredients to the framework. The preliminary results indicate that this can be determined empirically but this may not seem to be entirely satisfactory since this is a tedious and error prone process. It is however also the impression that experiments are absolutely crucial in order to obtain reliable models. We would therefore like to look into how to combine theory and practice in order to obtain as good models as possible using as few experiments as possible.

We do not think that there is a way around experimenting when determining the proper utilities for a task. This is however a quite manageable task but it still emphasizes the fact that it is necessary to build the system and try it out, i.e., purely theoretical studies are not feasible.

9.2 Learning of Models and Utilities

A possible way to reduce the amount of experimentation before getting the system running is to introduce learning. This also facilitates tracking of time varying conditions like, e.g., wear of the robot and the sensors and changes in the character of the environment. Such learning methods have been described by Lindner et. al. [8].

9.3 Sensitivity Analysis

When using models it is important to know how sensitive the performance of the system is to perturbations of the models. The highly non–linear nature of the decision problem makes it infeasible to make this analysis analytically and thus computer simulation studies on basis of real models should be performed.

9.4 Multiple Agents

The decision framework has here been applied to decide about sensing actions in a single agent. It would however be interesting to investigate how well the approach can be ported to systems with multiple, heterogeneous agents where the sensing is distributed so that one agent can do sensing for another one, if it better suited/equipped for that particular task. Also, the framework may be applied to task planning with multiple agents so that the decisions are not about sensor actions but rather about what agents should perform what sub–tasks. These are however perspectives that are well outside the scope of this project.

Appendix A

Examples of Description Files

In this appendix three description files for sensors, tasks, and module models are presented. The contents of the description files should be compared with the more formal explanation given in section 6.

The files are commented as found necessary but in general the format of the files is quite alike the syntax for `yacc` files. This style has been adapted because it allows for relatively abstract descriptions to be supplemented with C–code where necessary.

To ease the understanding of the examples a short explanation of the *yacc*–style format is required. Each file has 3 sections, the preamble, the main body, and the functions part. The sections are separated by a **%%**–marker.

The preamble contains two things: C–code header information (enclosed in %{ and %}) necessary for compiling the functions part, and "declarations" of variables in the main body. These declarations have the format '`%id var var var....`'.

The main body contains the descriptions of either the sensors, the tasks, or the module models. This description is in general a high–level symbolic description which however can make references to variables and C–functions. These referenced functions are then declared in the last part of the file, the functions part.

Thus, the description files contain both abstract descriptions and old fashioned C–code. This duality may seem peculiar but it offers an opportunity to unite abstract descriptions with control and "ingenuity" that can only be obtained at a lower specification level.

The way the sensor planner uses this type of description is that it parses the high level description and picks out the C–code parts and compiles them into the planner. This offers an advantage in computational efficiency over approaches where all of the description is parsed and interpreted.

A.1 A Sensor Description

This is an example of a sensor description file. The variable `stime` is the variable that holds the length of the time interval a sensor is requested for. This variable is in general input to the

sensor cost function. It is seen that this cost function just returns the value of **stime** which means that the requested time is used as the cost.

Furthermore, it can be seen that there are 3 tokens for the sonar while the other sensors only have one. This indicates that the vehicle and the line striper are unsharable while the sonars can be shared by 3 modules.

```
%{

#include <stdio.h>
#include <vwhelix.h>
#include <common.h>
#include "planner.h"

%}

%%

# syntax for sensors is:
# sensor "name" { num_tokens "cost function"}

sensor "vehicle" { 1 "stime" }
sensor "sonars"  { 3 "sonar_cost(stime)" }
sensor "linestriper"  { 1 "stime" }

%%

/*********************************************************************/
/* sonar_cost() is a "dummy" function to illustrate the use of     */
/* functions in cost-function-declarations.                        */
/*********************************************************************/
static float sonar_cost(stime)
float stime;
{
    return stime;
} /* end of sonar_cost() */
```

A.2 A Task Description

In this section an example description of a task called *door_traverser* is given. The task can be performed by the purposive modules, *pinger, door_est,* and *beta_est.* The actuator actions (declared with the %aa statement in the preamble) that can complete the task are *go* and *abort.* The world states that influence the expected utility of completing the task are *pos_cor, pos_wrong, door_open,* and *door_closed.* These world states are defined in the preamble with the

`%ws` statement. Δ for the process is 0.05 but just to illustrate that this can also be a function, the function `dt_delta()` has been defined.

The last part of the task description is the utility table relating actuator actions and world states. Each entry in the table is in principle a function, but in this case all but one of them are constants. The last one (which is the first entry) however illustrates how state information can be used to calculate the utilities. In this case the utility of going through the door given that the position of the door is known correctly is scaled with the probability that the door is open since this is an independent condition that has to be met in order to successfully pass the door. The input variable *task_data* is a pointer to a struct that holds relevant information about the current task (such as what door is currently being traversed).

```
%{

#include <ev_modules.h>
#include <aactions.h>
#include <world_states.h>

%}

%aa go abort
%ws pos_cor pos_wrong door_open door_closed

%%

task "door_traverser" {
    modules { pinger door_est beta_est }
    actuator_actions { go abort }
    world_states { pos_cor pos_wrong door_open door_closed }
    delta = "dt_delta()"
    utilities {
        U( go , pos_cor ) = "80.0 * Prob_Door_Open(task_data)"
        U( go , pos_wrong ) = "-80.0"
        U( abort , pos_cor ) = "-20.0"
        U( abort , pos_wrong ) = "10.0"
        U( go , door_open ) = "15.0"
        U( go , door_closed ) = "-30.0"
        U( abort , door_open ) = "-15.0"
        U( abort , door_closed ) = "15.0"
    }
}

%%

/*********************** delta stuff ******************************/

/*****************************************************************/
/* dt_delta() is a "dummy" function to illustrate the use of     */
```

```
/* functions in delta-declarations.                                    */
/*********************************************************************/
static float dt_delta(void)
{

    return 0.05;

} /* end of dt_delta() */

/************************ util stuff ******************************/

/*********************************************************************/
/* prob_door_open() returns the probability that the door (of      */
/* current concern) is open. This is used to scale the utility of  */
/* finding the position of the door.                                 */
/*********************************************************************/
static float Prob_Door_Open(task_data)
struct event_record *task_data;
{
float apriori[NUM_WS];
int ws;

    for (ws = 0; ws < NUM_WS; ws++) apriori[ws] = -1.0;
    apriori[door_open] = 1.0;
    Get_Apriori(apriori, task_data);

    return apriori[door_open];

} /* end of prob_door_open() */
```

All tasks should be described inside the same description file, but for brevity only one has been shown here.

A.3 A Module Model Description

In this section a description of a module model called *betaest_linestriper* is presented. Normally there would be more module descriptions in the file, but only one has been presented for brevity.

The module can generate two reports, *shut* and *up*, respectively. These reports are declared in the preamble with the **%r** statement. The corresponding world states that the reports testify about are *door_closed* and *door_open*. These states are declared in the task description.

However, the main part of the description is the probabilistic model of the module in form of a conditional probability table. This table defines how well the reports issued by the module reflect the true state of the world. This is in general a function of various state variables, in this case the uncertainty on the robot position. Understanding the code that calculates this function is not important in this context, only the principle behind it matters here.

```
%{

#include <stdio.h>
#include <vwhelix.h>
#include <world_states.h>
#include <mmodels.h>
#include <reports.h>

#define FALSE 0
#define TRUE  1

%}

%r shut up pos_ok pos_bad

%%

mmodel "betaest_linestriper" {
    reports { shut up }
    world_states { door_closed door_open }
    probs {
        P( shut | door_closed ) = "Pbetaest(door_closed)"
        P( up | door_closed ) = "1 - Pbetaest(door_closed)"
        P( shut | door_open ) = "Pbetaest(door_open)"
        P( up | door_open ) = "1 - Pbetaest(door_open)"
    }
}

%%

/*********************************************************************/
/* good_pos() returns TRUE if the vehicle uncertainty is low enough  */
/* for the line striper to see the door and not the wall.            */
/*********************************************************************/
int Good_Pos(event)
struct event_record *event;
{
int pos_good = TRUE;

    if (event->vehicle_status.unc.x > MAX_LS_POS_UNC) pos_good = FALSE;
    if (event->vehicle_status.unc.y > MAX_LS_POS_UNC) pos_good = FALSE;
    if (event->vehicle_status.unc.a > MAX_LS_ANG_UNC) pos_good = FALSE;

    return pos_good;

} /* end of good_pos() */

/*********************************************************************/
```

```
/* pbetaest() returns the probability of the beta estimator (line    */
/* striper) reporting "shut" based on world state info.              */
/*********************************************************************/
static float Pbetaest(ws)
int ws;
{
struct event_record event;
float res;

    if (ws == door_closed){
        /* if the door is closed the line striper reports shut no */
        /* matter what (since the wall looks like a closed door). */
        res = 0.95;
    }
    else {
        /* if the position uncertainty of the vehicle is less than */
        /* 20 cm the linestriper is very reliable. Otherwise it is */
        /* not since it does not "see" the door, but the wall, and */
        /* thus it always reports "shut".                          */
        Get_Status(GET_UNCERTAINTY, veh_driver, &event);
        if (Good_Pos(&event)){
            res = 0.02;
        }
        else {
            res = 0.95;
        }
    }

    return res;

} /* end of Pbetaest() */
```

Appendix B

The Door Finder

B.1 Introduction

As a part of a navigation system for a mobile robot it is instrumental to have a purposive module that can estimate the position of the robot with respect to the world coordinate system. The door finder is such a module specialised for estimating the position of a mobile robot with respect to a doorway. The doorways are assumed to be included in an a priori model of the environment and thus their position are known up to the precision of the model. The uncertainty in the world model can however be too large for relying on the model when traversing doors and therefore it is practical to estimate the position of the robot relatively to the door to be traversed when doing position estimation in connection with door traversal. This is what the door finder is intended for but at the same time it can be used as a general landmark recognition module.

The module uses an active sonar sensing strategy to do the landmark recognition.

In the following sections, the underlying assumptions for the door finder will be outlined, and an algorithm/architecture for it will proposed.

B.2 Basic Considerations and Assumptions

As described in the Introduction, we would like to design a door finder for estimating the position of a given door with respect to the vehicle. We will make a number of assumptions about the environment in which the door finder should operate. These assumptions in general reflect the fact that we are taking a model based approach, i.e., the system possesses a priori knowledge about the environment and it has a notion of what a door is. This is not considered to be un–realistic since a navigation system that utilizes a door finder necessarily must have at least a notion of "a door" and therefore probably also of a "non–door", i.e., a wall. Furthermore, we believe that providing navigation systems for man made environments with a priori knowledge generally is beneficial.

The assumptions underlying the design of the door finder are the following:

- The position of the door opening is known with bounded uncertainty.
- The position of adjacent walls are known with bounded uncertainty.
- The representation of door openings and walls is consistent, i.e., door openings and walls are mutually exclusive.
- The width of the door opening is known "exactly", i.e., with an accuracy that is relatively so much smaller than the resolution of the sensors that the unavoidable uncertainty is irrelevant.
- The position of each sensor at any time is known with uncertainty.
- A model of the sensors exists.

Apart from that there are of course many implicit assumptions such as the sensors are working, that nobody will tease the robot by walking between the sensors and the door etc. In general the above assumptions mean that the model is consistent, that it is known how the door "looks" and behaves, but that the relative position of the door opening with respect to the robot is only known to some extent.

The physical environment in which the system will be developed and tested is the GRASP Lab at University of Pennsylvania. This is a typical, newer office/laboratory environment.

The robot used will be a TRC Labmate base with 16 fixed sonars scanning a range of 200 degrees in front of the robot.

For the design of the door finder, we will adapt (and modify) the notion of a *feature detector* as defined in [3].

> [A] feature detector is realized as a control process that directs the robot's movement and sensing. On the basis of the data gathered during execution of a given feature detector, a probability distribution is determined for the proposition that the feature is present at a specific location.

In this case, we will not question whether the door is there or not, so rather than calling our system a "detector" we will call it a "finder" and the probability distribution will express only where the vehicle is with respect to the door, i.e., the probability distribution affected by the door finder is the uncertainty on the robot position.

The important thing to notice in the above definition is, however, that the door finder is a *control process*, i.e., it is something active capable of altering the state of the robot and the sensors, in this case their position and direction. Using this as the basis for our door finder means that we can do active sensing with the sonars although they themselves are not configurable (like, e.g., a stereo camera head)[1]. This ability is considered essential is this endeavor since it due to the

[1]To avoid confusion: Sonars are all active in the sense that they emit a signal to "illuminate" the scene just like a laser range finder is considered active because it sweeps its laser beam over the scene. In this context we will however use the term "an active sensor" in the meaning that it can have its sensing parameters configured to yield data well suited for the current task.

inherent functionality of sonars and the task at hand is doubtful whether passive sensing would be useful at all.

While the active behaviour potentially allows the system to come up with better results it also poses some additional problems. The most important of these is: How to be active? In this case this is the same as asking which strategy should be used to move the sensors and thus the robot around so that data facilitating a good estimate of the door position can be gathered.

We can divide sensing strategies up into two classes: fixed and flexible. Using a fixed sensing strategy means that the sensors are moved in the same way in each instance and thus no feedback path from the sensing to the sensor control exists. This is not the case when using a flexible sensing strategy, according to which the sensing is planned on–line based on the sensor data acquired so far and, e.g., physical constraints stemming from the scene.

Pros and cons for a fixed sensing scheme are the following:

- The "same" measurements are made every time thus providing better predictability of the result.
- Sensor data can be processed off-line.
- No on–line path planning is required.
- If pre–planned movement not possible due to obstacles, then what?

while they for a flexible sensing scheme are:

- An "optimal" movement can be obtained, i.e., maximal benefit from the active sensing scheme can be obtained.
- Sensor data must be processed on–line.
- Sensing can be stopped when sufficient data have been acquired.
- On–line path planning is required.
- A strategy as such is needed, i.e., some behaviour guiding the robot according to sensor input must be designed.

Since both sensing strategies have their advantages and drawbacks we suggest a hybrid approach that will try to use the best of both worlds. Basically what we suggest is a semi–fixed strategy with on–line processing. With a semi–fixed strategy is understood that one of several fixed movement patterns is traversed as far as the environment allows for this and as far as it is needed for obtaining a sufficiently good estimate of the door position. What movement pattern to chose will depend on the door environment. This means that no path planning is needed, only path checks. Starting out with using on–line processing means that if it should turn out to be necessary to introduce a flexible sensing strategy, the work done on the processing is not wasted.

The major drawback of the semi–fixed approach is of course that the sensing, i.e., the movement of the robot, can not be adapted to the sensor inputs. However, it is anticipated that the actual freedom the robot will have in a real sensing situation will be limited because of the physical constraints and because the robot will have to stay quite close to the door opening to obtain useful data with its sonars. The demand that the data processing is done on–line is assumed to be realistic, since this work will build on the work by Robert Mandelbaum [9] in which the processing is done on–line.

B.3 The Algorithm

In this section the overall algorithm proposed for estimating the door position is outlined. In the algorithm it is assumed that the vehicle starts out from a "known" position in front of the door. The proposed algorithm is as follows:

select pre–planned movement from library
start sensing and sensor data processing
while not end of movement
    check pre–planned path
    move along pre–planned path
    process sonar data
estimate position of door posts
estimate position of vehicle

The first step in the algorithm is to select a pre–planned path from a path library according to features of the door that is currently in question. Then the sensing and data processing is turned on and the robot traverses the part of the pre–planned path that is possible. The data processing itself follows the algorithm outlined below:

estimate line segments from sensor data
match estimated line segments against model data (walls)
if match obtained
    update vehicle position
update door post map

The estimation of line segments from sensor data is based on the work by Mandelbaum and is described in [9]. However, this algorithm has been modified into being more model–based so that it only looks for line segments where walls are expected to be found. When such line segments have been estimated they are matched to the model walls. This basically should fix the orientation of the robot and the position in at least one direction (depending on the wall configuration). This is illustrated in figure B.1.

If a "sufficiently good" match between the estimated walls and the model walls is obtained, the algorithm updates the position of the vehicle. Then it updates the map in which it searches for the door posts. This is done after matching the walls in order to compensate for movement errors. The door post map is a special grid which is updated with the sonar readings.

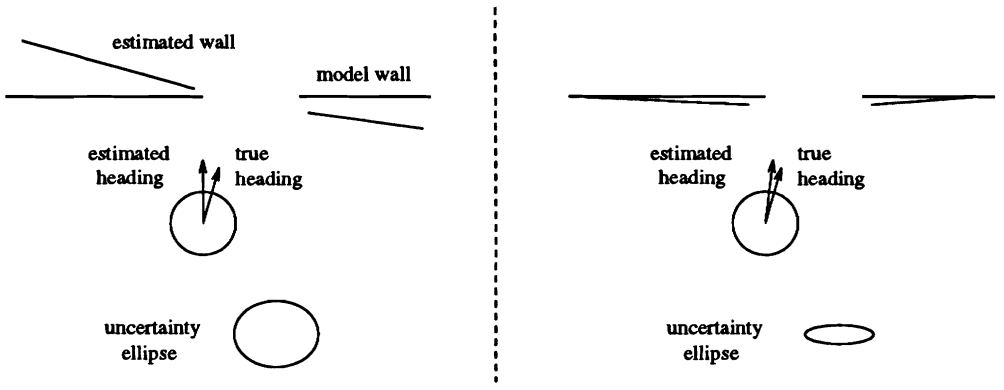


Figure B.1: *Illustration of uncertainty on robot position and orientation when matching estimated walls against model walls. Left is situation before matching and right is after matching. If there had been any "vertical" walls, the uncertainty in the "horizontal" direction could also have been reduced significantly (depending on the quality of the match). The uncertainty ellipse is really an ellipsoid where the third dimension represents uncertainty on the orientation. This, however, is not illustrated.*

The fact that is utilized in the door post map is that the door frame has corners and thus works as a corner reflector for the acoustic waves. This means that, as opposed to a planar surface, the sonar signals can be reflected in other directions than normal incidence. Thus, corners can be detected by registering how many times a point in space has reflected a sonar beam with different incidence angles. In the door post map, which is basically a histogram, each sonar reading in the vicinity of the expected position of the door posts is recorded as an arc, where each cell on the arc can be considered as a hypothesis of a reflecting point. With each cell in the map is associated a list of angles indicating from which directions the point has been detected. The cell count is then only updated if the arc intersects the cell and if the incidence angle differs by more than $\Delta\alpha$ from previously recorded intersections. A map from a typical experiment is shown in figure B.2

It is clear that this strategy can only be used with an active sensing strategy since otherwise points would normally only be detected from one angle.

The door posts are found from the maps by first creating hypothesises by detecting maxima and then matching the door post hypothesises from each map. The matching process is model driven in the sense that a match is sought that satisfies the constraint on the door width given by the model. If no significant maxima are found or if no match can be found that satisfies the model then the vehicle position is not updated, otherwise the vehicle position is updated using model information about the door and the relative door post position found from the maps.

B.4 Pre–Planned Movements

In this section we will discuss how to implement the semi–fixed sensing strategy. This especially means how to make the robot move in a semi–fixed way and how to determine what movements would be optimal.

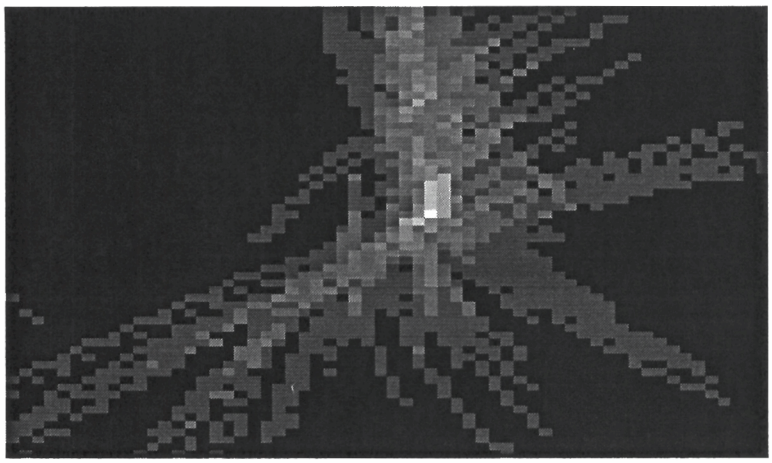


Figure B.2: *Example of door post map. White denotes a high histogram count, dark a low count. The histogram was scaled for illustration purposes. The cell counts were in the interval [0;8]. In the map, a door post can be recognised approximately in the center and the adjacent wall extends upwards out of the map. A map is generated for each door post.*

The whole point of using pre–planned movements is to avoid on–line path planning and instead only do path checks with, e.g., a sonar bumper. However, in most real word scenes it is questionable whether such a pre–planned path can often be traversed completely. In order not to have to abort the whole "mission" just because part of the path is not traversable, the path should be divided up into sub–paths connecting set–points that the robot should move between. If part of the path then can not be traversed, the next set–point should be selected as the current goal point. So far, we will consider straight paths between set–points only. Parameters for each set–point will be whether the robot should move forwards or backwards to reach it and what direction the robot should face after reaching the set–point. Rotations on the spot can thus be implemented by having two coincident set–points with different angles. An example of a pre–planned path is illustrated in figure B.3.

B.5 Experiments

To verify the functionality of the module, a number of experiments were carried out. Unfortunately, experiments numerous and systematically enough for reliably modelling the behaviour of the module were not conducted. It was however found that the door finder normally performed as expected and determined the position of the vehicle with an error of less than 10 cm. However, as the uncertainty on the vehicle position was increased, the performance decreased until no useful results were obtained at position uncertainties larger than 50 cm. This is due to the fact that the movement strategies do not enable the robot to sense the door posts for such large displacements. Also, due the the realistic test environment, other doors/corner reflectors were detected when the vehicle was far off its believed position.

On basis of these somewhat subjective findings a probabilistic model of the door finder was created for the test of the whole sensor planning system. This model is given in table B.1 and figure B.4. The model expresses the probabilities of receiving the reports *pos_ok* and *pos_bad*, i.e., that the module thinks it has/has not found the door position, given that the real situation

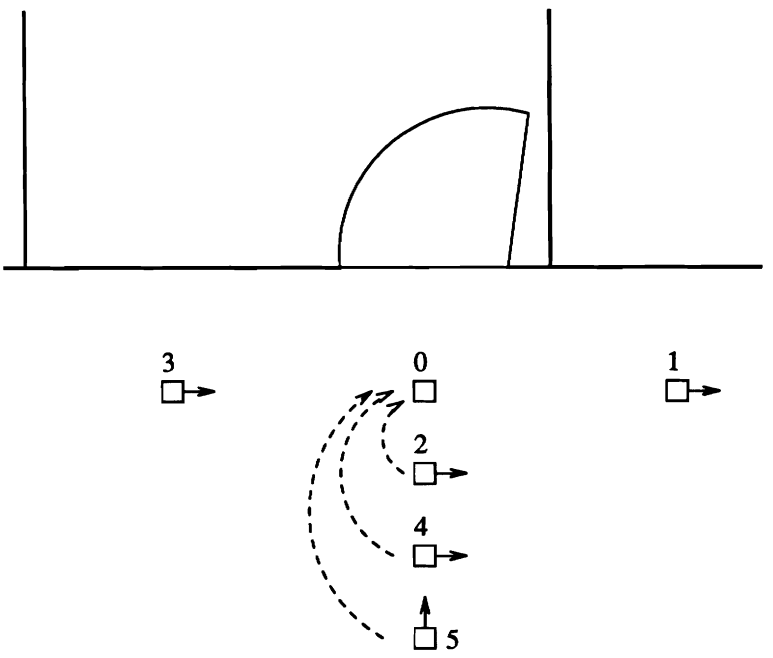


Figure B.3: *An example of a pre-planned path for estimating the door position for a door that opens away from the robot. The set-points are shown as small boxes with a number on it and a direction indicating how the robot should turn when it gets there. The 4 set-points in the middle are all at the same location, namely that of point no. 0. This path will take the robot forth and back in front of the door, allowing for the sensors to sense the walls and the door posts. The robot will end up facing the door opening.*

	pos_cor	pos_wrong
pos_ok	0.85	$P(pos\_ok \mid pos\_wrong)$
pos_bad	0.15	$1 - P(pos\_ok \mid pos\_wrong)$

Table B.1: *The probabilistic model of the door finder. Refer to figure B.4 for* $P(pos\_ok \mid pos\_wrong)$.

after the door finder has been run is *pos_cor* and *pos_wrong* respectively, i.e., that the position of the robot is actually correctly/wrongly estimated. The position is considered to be correctly estimated if the error on the vehicle position is less than 10 cm. This corresponds to the quantity e_{max} discussed in section 4.5.

From the model it is seen that if the robot's position is really known, the door finder will normally (in 85% of the cases) find the door and in the remaining 15% of the cases it will not. If the robot position is however not correct the success-rate depends on how far off it might be. This is described by the uncertainty on the robot position and thus $P(pos\_ok \mid pos\_wrong)$ is a function of the vehicle uncertainty. This function is illustrated in figure B.4 and basically just illustrates that the probability of the module getting fooled (reporting that it has found the door when it has really not) increases as a function of the vehicle uncertainty. The actual values in the function are assumed to be somewhat conservative but this has been preferred because of the lack of systematic experiments.

The time spent performing the door finding was on the average 100 seconds. This is quite long

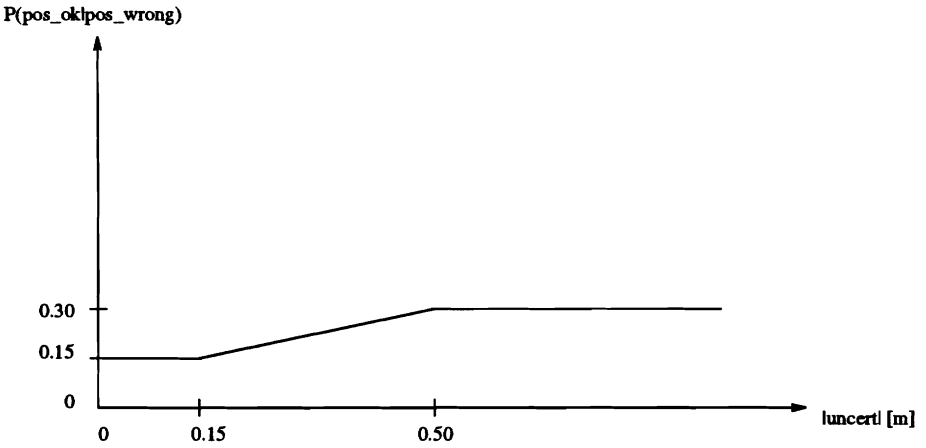


Figure B.4: *P(pos_ok|pos_wrong) as a function of the vehicle uncertainty. See text for further explanations.*

time which of course is due to the active sensing strategy which makes it necessary for the robot to move around quite a lot. This means that the door finder is a quite expensive module to execute.

B.6 Conclusion

A purposive module for locating doorways using an active sonar sensing strategy was designed and tested.

The door finder used a model based approach to locate doorways and the surrounding walls. This and the active sensing strategy resulted in quite robust performance but unfortunately also in a rather expensive to execute module.

Appendix C

The β Estimator

In this chapter a purposive module for estimating the angle, β, of a door relative to the door frame is described. The module is intended for mobile robot navigation and uses a line striper and a CCD camera as sensors.

C.1 Introduction

When a mobile robot is about to pass through a doorway it should estimate the state of the door in order to decide whether the doorway is traversable or not. With traversable is understood that the door is open enough for the robot to pass. The robot used for the navigation experiments is equipped with a line striper/CCD camera constellation (in the following just called "the line striper"). This seems to be a appropriate sensor for estimating the door angle, i.e., the angle between the door and the door frame since this sensor provides sparse, high resolution output which is exactly what is needed to estimate a single parameter, the door angle, robustly and accurately.

Since this module might not be the only one to use the line striper the module has to get permission from the sensor planner to use it. This implies that the module should be modeled to facilitate the planner with means for deciding when to grant the module the line striper.

In the following sections, the geometry of the problem will be described, the algorithms developed will be explained, and test results will be presented. Finally, some concluding remarks are made.

C.2 Estimating Angles with a Line Striper

When estimating the door angle with the line striper, a two–stage process will be employed. The first step is to extract a set of 3D points and the second step is from this set to calculate the slant of the surface passing through the points. The reason for using this two stage process in which 3D points are explicitly extracted is that it is intuitively appealing and that the line striper can more easily be replaced with another 3D point sensor.

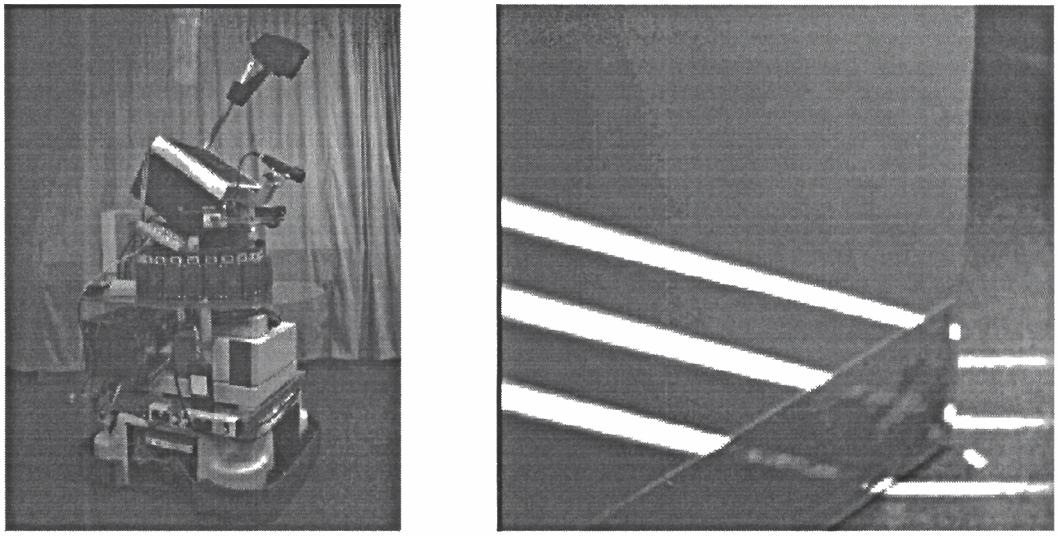


Figure C.1: **Left:** *The mobile robot with the line striper. The line striper is the thing on the top that resembles an overhead projector. This is not a coincidence.* **Right:** *A typical image recorded by the line striper CCD camera showing the light stripes projected onto a partly opened door. Normally, the aperture of the camera would be smaller, so that only the light stripes would be visible in the image.*

The majority of this section will be about the line striper geometry used to extract the 3D points, since the second step—the calculation of the slant, is relatively trivial.

The robot with the line striper (which *could* be mistaken for an overhead projector) is shown in figure C.1. Here is also shown a typical image as recorded by the line striper CCD camera.

For simplicity and ease of illustration we will initially consider a point light source (as e.g., a laser) instead of a light line source as the line striper. Since we are working with discretely sampled signals anyway we can consider the light stripe as a number of points without loss of generality.

The geometry associated with using the line striper as a depth sensor is illustrated in figure C.2.

In figure C.2 the optical axis and its normal define a coordinate system with origin in the optical center of the camera which is assumed to have pinhole characteristics. B is the baseline of the sensor, defined as the separation between the the optical axis and the axis of the light source measured along the x–axis of the camera coordinate system. From this it can be seen that

$$x' = z' \tan\alpha \tag{C.1}$$

and

$$x' - B = z' \tan\gamma \tag{C.2}$$

thus, in general

$$z(\tan\alpha - \tan\gamma) = B \Rightarrow z = \frac{B}{\tan\alpha - \tan\gamma} \tag{C.3}$$

where B and γ must be calibrated and α extracted from the image. This, however turns out to

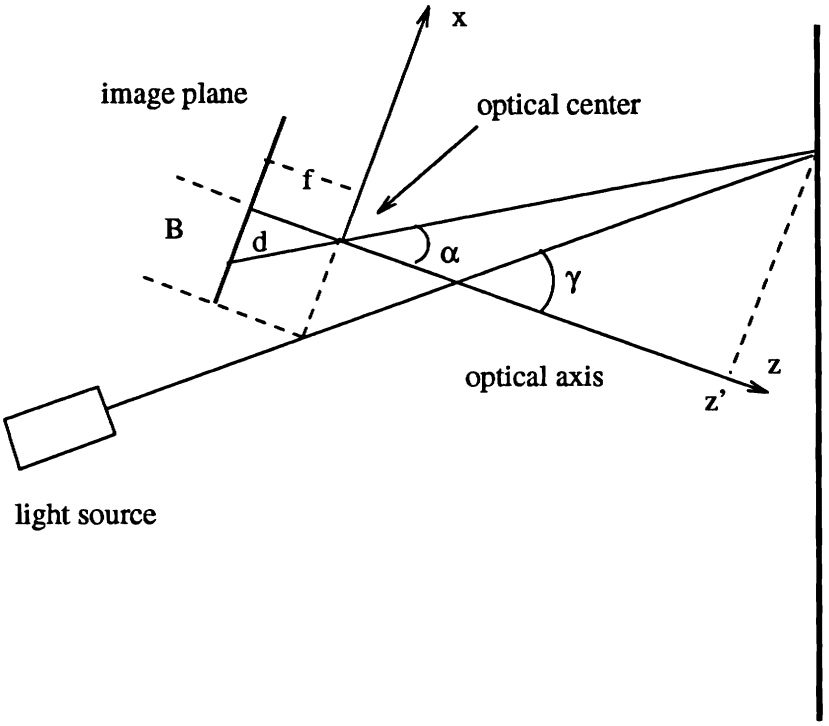


Figure C.2: *The geometry of using a light point source and a camera as a depth sensor using triangulation. See text for further explanation.*

be quite straightforward since it is seen that

$$\tan\alpha = \frac{-d_x}{f} \tag{C.4}$$

where f and d_x are the focal length and the coordinate of the light point image, respectively. These two quantities have to be measured in a common unit where the easiest and most natural is pixels. From equation C.3 and C.4 we can summarize that

$$z = \frac{B}{\frac{-d_x}{f} - \tan\gamma} = \frac{-B}{\frac{d_x}{f} + \tan\gamma} \tag{C.5}$$

where B will be a negative quantity. The corresponding x and y coordinate can the be found as

$$x = \frac{d_x z}{f} \text{ and } y = \frac{d_y z}{f} \tag{C.6}$$

Now, all this is in the camera coordinate system. In general, to transform this into world coordinates, a transformation matrix must be multiplied with each point. This is however not necessary since we assume that the line striper is only tilted with respect to a camera centered world coordinate system (which is rotation around the camera coordinate system y–axis pointing out of the paper in figure C.2). This means that the y–coordinates are unchanged, i.e., $Y = y$, and that the depth in world coordinates is related to the depth in camera coordinates by a cosine, i.e., $Z = z\cos\phi$, where ϕ is the angle the camera is tilted with respect to horizontal. By analogy $X = x\cos\phi$. This however only transforms the points into a camera centered world coordinate

system. The transformation into robot or world centered coordinate systems is straightforward, and although it is relevant for our objective (since the position of the door frame is given in world coordinates) we will not pursue this topic further here.

Now all that is left is to calculate the slant of the reflecting surface. In order to do this, we must know at least two non coincident 3D points from the surface (two is enough since we assume that the surface is vertical—which however means that the points may not be located on a vertical line). Since the real sensor, the line striper, contains a light stripe source and not a light point source this criteria is easily fulfilled, and thus the door angle, β, can be found as:

$$\beta = \arctan \frac{Y_2 - Y_1}{Z_2 - Z_1} \tag{C.7}$$

It is critical that the points (d_x, d_y) used to calculate (Z, Y) are selected carefully to provide a robust and accurate estimate of β. This is the major topic of the next section.

C.3 Algorithms and Implementation

Since the mathematics of the door angle estimation is relatively simple, the main concern in the implementation of the module has been put on getting a robust and accurate estimate.

The overall algorithm of the module is as follows:

move line striper into position in front of door
grab image
extract line segments from image
for each line segment
    calculate door angle
determine final door angle

In the following, the positioning and the extraction of line segments will be described more in detail.

Move Line Striper Into Position

Since the line striper is rigidly fixed to the robot it is necessary to move the vehicle in such a way that the line striper is aimed towards the doorway of interest. Also we want the line striper to be directed so that the door can be seen no matter what angle it is in. In other words we must use an active perception strategy to use the line striper for estimating β. The desired robot/door configuration is illustrated in figure C.3.

The necessary setpoint and configuration angle can be calculated from model data and (dt, dn) which is the displacement of the robot in the tangential and normal direction of the door way with respect to the door center. The robot is turned to an angle that should direct the line striper towards the center of the door way. Because the position and the angle of the robot is

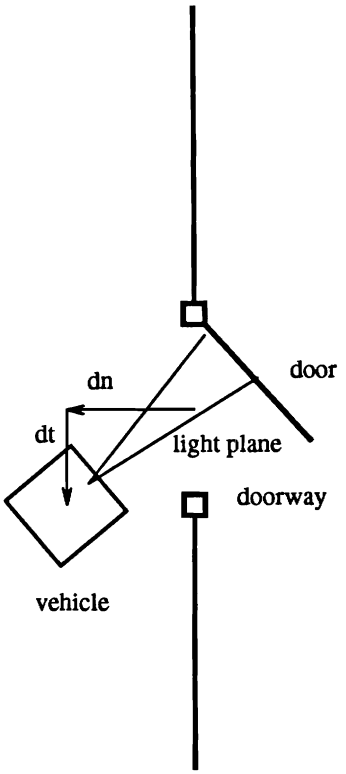


Figure C.3: *Configuration of line striper relative to door way. No matter which angle the door is in it can be picked up by the line striper.*

uncertain, it is however important that *dt* and *dn* are chosen so that the line striper "sees" the door although its position is not perfect. This is also illustrated in figure C.3 where the robot angle is offset so that it does not aim towards the center of the door. While moving to the setpoint, the robot checks the path with its sonars. This is necessary because no on–line path planning is used for the movement since normally the robot only has to move a small distance. If the sonars report an obstacle in the way of the robot, the estimation of the door angle is aborted.

It is however important to notice that the fact that the robot has to be moved prior to the measurement means that not only should the module be granted control of the line striper but also the robot base and the sonars need to be available to the module.

Extraction of Line Segments

To get some point sets, (d_x, d_y), from which to estimate β it is necessary to extract the light stripes in the image projected by the light source.

The light source itself projects three parallel, horizontal stripes onto the environment (see figure C.1). Due to discontinuities in the environment these stripes may be broken up into more segments. One way to proceed would be to use all segments from all three stripes for estimating a number of β–values which then, e.g., could be averaged to obtain a final value. However, since there is much more information in just one stripe than we need (we only need two points)

two of the lines will instead be used for making the estimation more robust and we will then concentrate on the center line. This also reduces the need for calibration since only one angle, γ, needs to be determined. The sequence of finding the line segments of interest is:

detect (seed) points on center stripe
grow regions from seed points
for all regions
    extract line points from region
    fit line to points
    (estimate β)

The philosophy behind the algorithm is basically to extract all pixels belonging to the center stripe using simple region growing. It has been chosen to extract all pixels (the "line" is not a one–pixel thick line in the image) belonging to the line to get maximal statistical robustness. This is efficiently done with a queue–based region growing algorithm if the seed points are selected robustly. To do this, it has been exploited that the stripe of interest is the middle of three stripes. Thus, to select seed points for the region growing, every N'th column of the image is scanned, and if exactly three maxima over a given threshold are detected, the center point of the center maxima is selected as a seed point. This results in M seed points from each of which (in sequence) a region growing is started. If there is only one line segment from the center stripe in the image, only the first region growing will result in a region (containing all pixels from the center stripe above a threshold) since the rest of the seed points will be contained in this region. Likewise, if the center stripe is broken up into K segments, the region growing will result in K regions. However, due to the positioning strategy, normally only one region should result.

Now for each region the centroid of the belonging pixels in each column is determined with sub–pixel accuracy. This results in a set of pixels constituting the "skeleton" of the region. This should ideally be a line but since this is not the case in the real world, a line is fitted to this set of points using standard linear regression. Thus, the end result of the process is a "mathematical" line for each region. From each such line it is then easy to extract two points, (x_d, y_d), from which β can be calculated using equations C.5 and C.7.

If more than one line segment is detected a way to determine the final value of β must be determined. Since it really does not make any sense to average the values from each of the segments it has instead been chosen to select the β–value stemming from the segment with most support in the image, which in this case is interpreted as the line with the largest horizontal extent.

C.4 Test of β–Estimation

In this section the results of a number of tests conducted with the line striper module are reported. The purpose of the tests is to prove that the algorithms described in the previous section work but most importantly the purpose is also to form a basis for estimating *how well* they work.

In the first test, the vehicle was placed in front of a door in an "optimal" position, i.e., a position that corresponds to the case where the initial robot position is known with high accuracy. From

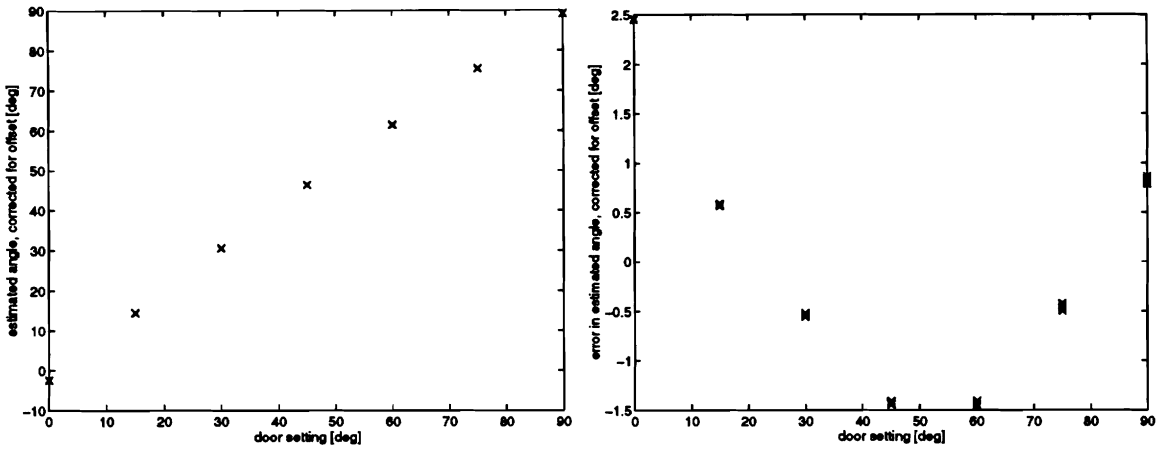


Figure C.4: **Left:** *The result of measuring each angle 10 times while not moving nor the door the robot in-between measurements. All 10 measurements for each door angle are plotted.* **Right:** *The error between expected and measured angle. It is seen that there seems to be a systematic error.*

that position, the vehicle was driven to its "setpoint" as indicated in figure C.3. During the experiments the robot was stationary and only the angle of the door was changed. The angle of the door was increased with steps of 15 degrees in the interval from 0 to 90 degrees. In each position, 10 measurements were made. The results are shown in figure C.4. In the plot the data have been corrected by the offset due to the fact that the robot is not perpendicular to the door when $\beta = 0$ degrees. This corresponds to transforming the measurements from a robot centered coordinate system to a world centered coordinate system. The offset has been calculated from the data with the assumption that the error on the measurements has zero mean. This need probably not be true, but this will show in later experiments. In figure C.4 the error between the expected and the measured data is also plotted.

From figure C.4 it can be seen that the variance on the measurements is small but that there seems to be a systematic error. Where this error comes from is not clear but the most probable explanation is that it stems from calibration errors and the fact the the CCD camera does not have ideal pinhole characteristics.

The error could also stem from the positioning of the door, but it is believed that the position can be set more accurately than 1 degree. Furthermore, a second test where the door but not the robot was moved between measurements showed that the door can be re–positioned with a deviation of less than 0.23 degrees. This figure is important to keep in mind for the subsequent tests, as it can be considered as a kind of "background" noise that cannot be eliminated (the "noise level" will however change discretely, namely each time the door is moved to a new position).

In the second experiment the robot was before each measurement driven back to its outset right in front of the door. The outset position was marked and the robot placed there to the best of the experimenter's abilities. It is estimated that the start position of the robot each time was within a 2 cm × 2 cm square and that the orientation of the vehicle was within ± 2 degrees. For each setting of the door (that was not moved between experiments with same β) 5 measurements were taken. The results are shown in figure C.5. In this experiment the angle offset was not

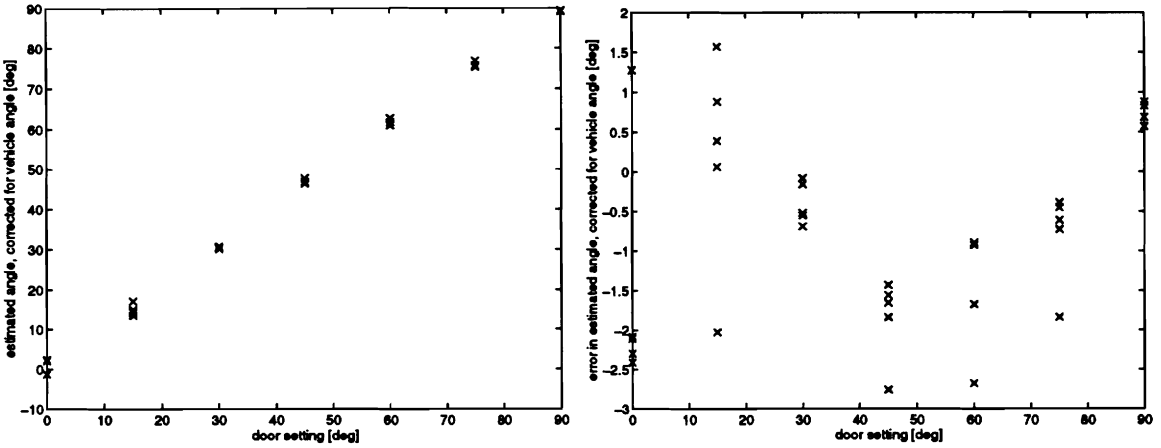


Figure C.5: **Left:** *The result of measuring each angle 5 times while moving the robot but not the door in-between measurements. All 5 measurements for each door angle are plotted.* **Right:** *The error between expected and measured angle. It is seen that there seems to be a systematic error resembling the one in figure C.4.*

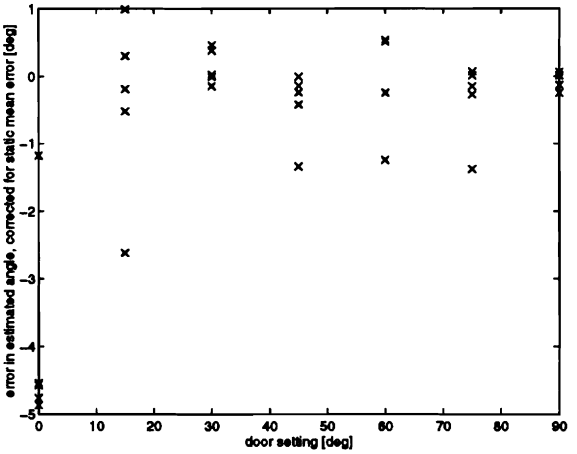


Figure C.6: *The error from figure C.5 equalised with the mean error from figure C.4.*

calculated from the data but rather measured with the vehicle's odometry. It is seen that the variance on the measurements is considerably larger than in the first experiment. The excess variance is stemming from the inaccuracies in the robot's movements/odometry. It is however also seen that the error exhibits the same pattern as was the case for the first experiment. To investigate this further, the mean of the error for each angle in figure C.4 was calculated and subtracted from the error in the plot in figure C.5. The result is shown in figure C.6.

From figure C.6 it is seen that subtraction of the "stationary" error has the effect of equalising or "whitening" the error on the experiment where the vehicle was moving. The only exception is the case for $\beta = 0$ degrees. Why this is an exception is unclear but a possible explanation is that these were the first experiments to be made with the robot moving and thus the experimenter may not have been as "stable" as in the subsequent experiments. Furthermore, the vehicle is suspected to have certain warm–up phenomena that could also cause differences in the movement patters in the start of the experiment.

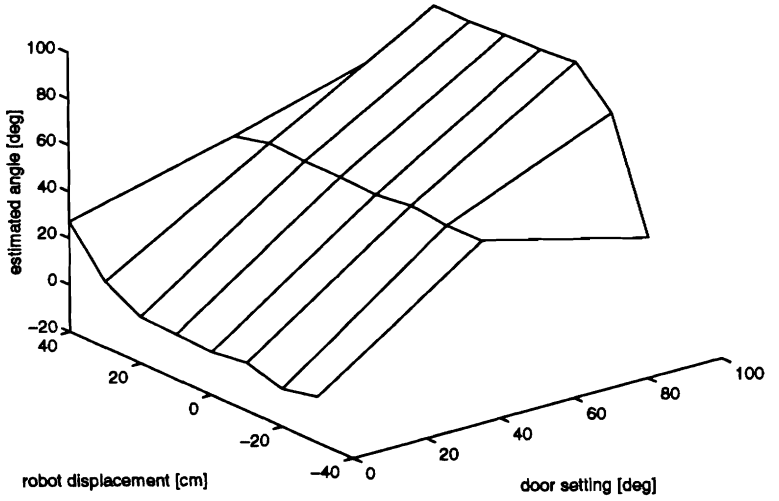


Figure C.7: *Estimate of β as a function of true door angle and vehicle position error.*

In the final set of experiments the robot was systematically displaced from its believed outset position in order to investigate the effects of position error on the results. To reduce the number of experiments, the robot was only displaced along the tangential direction of the door. 5 experiments were conducted for each value of displacement (in the interval $[-40cm; 30cm]$) and for each displacement, the door was positioned at 0, 45, and 90 degrees. The mean of the results is displayed in figure C.7 and the error is shown in figure C.8.

From the plots it can be seen that the error is quite small as long as the displacement is less than 30 cm. Then, however, the error rapidly increases. This non–linear behaviour is due to the similar non–linearity of the employed algorithm where β is estimated from the longest line segment and not calculated as, e.g., an average. This means that as long as the majority of the seen line is projected at the door, β is estimated from the correct data and thus rather accurately. However, with displacements $\geq 30cm$ the line striper really sees the wall beside the door and thus estimates β to be close to 0 degrees (which explains why the error is small for the true door angle = 0 degrees).

An execution of the β estimator on the average takes 8 seconds which means that it is a relatively cheap module to use.

C.5 A Model of the β Estimator

On basis of the experiments described above, a simple probabilistic model was created of the β estimator. The model is shown in table C.1.

In the model it is reflected that the β estimator is quite reliable when the error on the robot position is less than 20 cm and that the modules "always" reports *shut* when the error is larger. Since the displacement of the robot is not known, but only the magnitude of the position uncertainty, this quantity has been used instead. This, however, makes the model somewhat

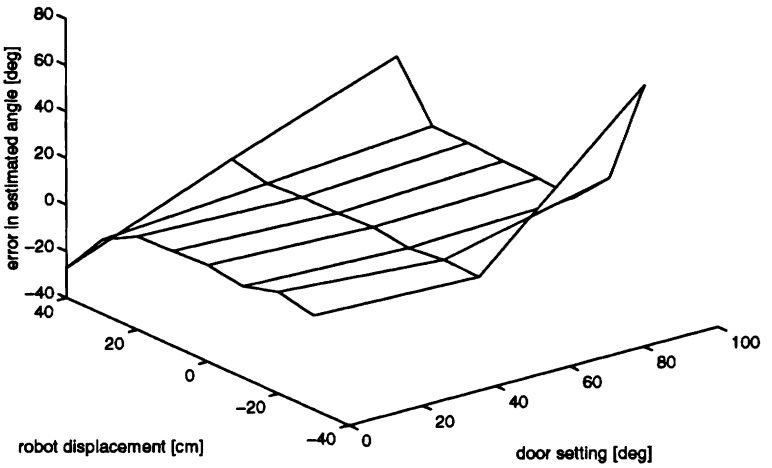


Figure C.8: *Error on the estimate of β as a function of true door angle and vehicle position error.*

	unc ≤ 20 cm		unc > 20 cm	
	door_closed	door_open	door_closed	door_open
shut	0.95	0.02	0.95	0.95
up	0.05	0.98	0.05	0.05

Table C.1: *The probabilistic model of the β estimator. Unc is the uncertainty on the vehicle position.*

conservative since the real position error will be ≤ the position uncertainty.

C.6 Conclusion

In this chapter a door angle estimator using a line striper was presented.

Emphasis was put on simple and robust algorithms which was made feasible due to an active sensing strategy optimizing the sensing conditions for the line striper. The algorithms were quite general and the β–estimator can thus quite easily be converted to other purposes.

Experiments showed that the β–estimator was able to estimate the door angle with a precision of about ±2 degrees. The effect of position errors was investigated and it turned out that as long as more than half the line stripe was projected onto the door the results were largely unaffected by the position error. This fact made it possible to derive a simple parametric expression for approximating the reliability of the module as a function of position error.

Appendix D

The Door Pinger

D.1 Introduction

The door pinger is a purposive module designed for detecting whether a door is possibly closed. The door pinger uses the simple strategy of firing a sonar through the door opening and waiting for a possible return signal to see if the door is closed. If a return signal (within given bounds) is detected it can be assumed that the door is closed. This strategy can however not tell if the door is open, i.e., if the angle, β, between the door and the door frame is large enough for the robot to pass, since the acoustic beam can be deflected even though β is relatively small. The advantage of the simple strategy is that it is very cheap (in sensing time and computational cost) and the aim of building the door pinger was to create a module that did not give complete results but which was cheap to use.

To reduce the load on the sensor planner, the door pinger was designed as a daemon in the sense that it runs "in the background" without requesting the sonars until an opportunity is there to fire a sonar through the door opening of current interest. This is detected by the door pinger from model data and by monitoring the vehicle position.

D.2 Algorithms

In this section the algorithms used in the door pinger are explained. The overall algorithm is as follows:

```
wait for startup command
get door model data
while no abort command
    monitor vehicle position
    if sonar in good position
      ask for sonars
      if sonars granted
        ping sonar
```

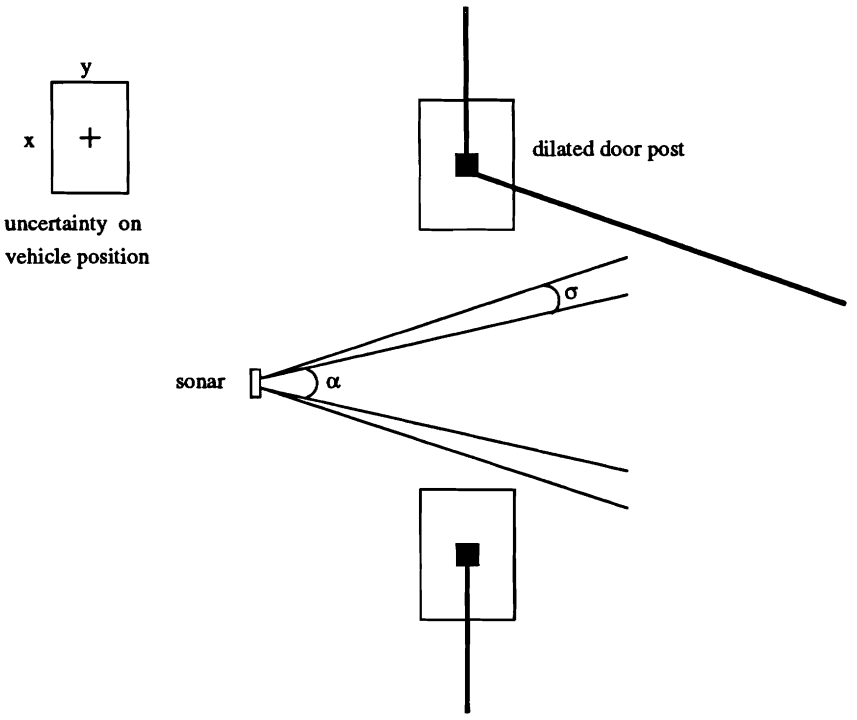


Figure D.1: *The door posts are dilated with the uncertainty (x,y) on the vehicle position. The sonar beam with opening angle α is expanded with the uncertainty on the vehicle position, σ.*

update door state

When the door pinger is told to start up it retrieves the model data of the relevant door (which is provided in the startup command). Then it starts monitoring the position of the vehicle including the uncertainty on the vehicle position. From this information, the door pinger can calculate whether a sonar is in a position allowing it to ping through the door. When calculating this, the inclusion of uncertainty information is important since the sonar *never* should detect the door frame of an open door instead of the door itself, i.e., we will not tolerate the confusion of a door frame and a closed door.

This can happen if the vehicle is not quite in the position it should be and thus it is important to also utilize uncertainty information. How this is done is illustrated in figure D.1 where it is shown how the door posts are dilated with the amount of uncertainty on the vehicle position.

The criterion for recognizing that a sonar is in a position to ping through a door is that the expanded sonar beam, which is the beam with the opening angle of the physical sensor plus the uncertainty on the vehicle orientation, does nowhere intersect the dilated door frame and that the beam goes between the door posts. If this is found to be the case for at least one sensor, then the sonars are requested from the sensor planner. If this succeeds then the sonars are fired and it is re–calculated if any of the sensors at the time of the reading was in a proper position. If this is the case then the return signal from the relevant sonars is investigated and if it is found that one of the sonars has got a return from an object at the approximate distance of the door (or closer), then the probability for that particular door being open, P_o, is updated to a low value,

	door_closed	door_open
shut	0.4	0.0
up	0.6	1.0

Table D.1: *The probabilistic model of the door pinger.*

P_p. If none of the relevant sonars got a return signal from the door then P_o is left unchanged. This may not be quite correct since the lack of a return signal may indicate an increase of P_o, but that increase would in any case be quite small and also depend on the angle of the sensor with respect to the doorway. Thus, it has been chosen not to update P_o when no proper return signal has been detected.

In the current implementation, the door pinger has built–in "modesty" in the sense that it stops itself when it has once succeeded to ping the sonars at the current door of interest. This has been done to save resources and to avoid a deadlock in the case that the door pinger wins the sonars each time but does not get a return signal and thus not changes any state variables that would enable other modules to win the sensors. Also, the door pinger is believed to be sufficiently robust for one sensing to be sufficient.

D.3 Experiments

In order to assess how well the algorithm worked a number of experiments were conducted where the door pinger was employed in front of a door which angle was varied. Due to the quite simple functionality of the module these experiments were not very numerous but the conclusion was clear—the door pinger consistently detected the door when the angle between the sensor and the door was ≤ 15 degrees and it never detected the door as closed when the angle between the sensor and the door was greater. There was no instance where a door post was detected as a closed door. This is reflected in the simple model presented in table D.1.

The execution of a "ping" takes less than 1 second.

D.4 Conclusion

A simple module for detecting the state of a door was designed and tested. The module was made as simple as possible to achieve cheap operation at the expense of incomplete results. This was desired as an alternative to more elaborate modules which provide complete results for a higher price.

It was found that the module operated as expected and could detect closed doors if the angle between the sensors and the door was ≤ 15 degrees. This was to be expected since the opening angle of the utilized sonars was 30 degrees (or ± 15 degrees). This, however, does not mean that the door pinger can detect all doors that are less than 15 degrees opened relatively to the door frame since the sensor is not necessarily aligned with the door frame. A possible improvement would be to include a criteria for sensor angle with respect to the door frame when determining if a sonar is in a proper position for sensing the door.

Bibliography

[1] Ruzena Bajcsy. Active perception. *Proceedings of the IEEE*, vol. 76(no. 8):996–1005, August 1988.

[2] H. I. Bozma and J. S. Duncan. A game–theoretic approach to integration of modules. *IEEE PAMI*, vol. 16(no. 11):1074–1086, November 1994.

[3] Thomas L. Dean and Michael P. Wellman. *Planning and Control.* The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, California, 1991.

[4] C. Giraud and B. Jouvencel. Sensor selection in a fusion process: a fuzzy approach. In *Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 599–606, Las Vegas, Nevada, October 1994. IEEE.

[5] Gregory D. Hager. *Task–Directed Sensor Fusion and Planning: A Computational Approach.* The Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, Boston, 1990.

[6] Finn Verner Jensen. *Introduction to Bayesian Networks.* Aalborg University, February 1994.

[7] Sukhan Lee and Xiaoming Zhao. Sensor planning with hierarchically distributed perception net. In *Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 591–598, Las Vegas, Nevada, October 1994. IEEE.

[8] John Lindner, Robin R. Murphy, and Elizabeth Nitz. Learning the expected utility of sensors and algorithms. In *Proceedings of the 1994 IEEE International Conference on Multisensor Fusion and Integration for Intelligent Systems*, pages 583–590, Las Vegas, Nevada, October 1994. IEEE.

[9] R. Mandelbaum and M. Mintz. Sonar signal processing using tangent clusters. In *Proceedings of the OCEANS '94: special session on Automated Unmanned Vehicles*, volume 2, pages 544–549, September 1994. Brest, France.

[10] Kristian G. Olesen. Causal probabilistic networks with both discrete and continuous variables. *IEEE PAMI*, vol. 15(no. 3):275–279, March 1993.

[11] Judea Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* The Morgan Kaufmann Series in Representation and Reasoning. Morgan Kaufmann, San Mateo, California, 1988.

[12] Howard Raiffa. *Decision Analysis: Introductory Lectures on Choices under Uncertainty.* The Addison–Wesley Series in Behavioral Science: Quantitative Methods. Addison–Wesley, Reading, Massachusetts, 1968.

[13] K. A. Tarabanis, P. K. Allen, and R. Y. Tsai. A survey of sensor planning in computer vision. *IEEE Transactions on Robotics and Automation*, vol. 11(no. 1):86–104, February 1995.

[14] S. Zilberstein and S. J. Russell. Anytime sensing, planning and action: A practical model for robot control. In *Proceedings of the 1993 Internation Joint Conference on Artificial Intellignece*, pages 1402–1407, Chambery, France, 1993. IJCAI.