Controlling Robot Arms: An EM Approach

Abstract

At first glance, the modelling of a robot arm might seem an easy task. Indeed, much effort in this area seems to involve the machine-vision problem of determining the position of the arm with relation to the object of our interest. Once this problem of 'target location' has been solved, the problem remains of moving the arm to that target position. This 'secondary' problem of arm control is examined in this paper. Empirical Modelling offers us the opportunity to explore the control of a robot arm in the 'ad-hoc' fashion, allowing various techniques to be tested in various situations, and to evaluate and compare their effectiveness. This paper also discusses possible enhancement of the current EM tools, (re)introducing a notion of object-enriched empirical modelling.

1 Introduction

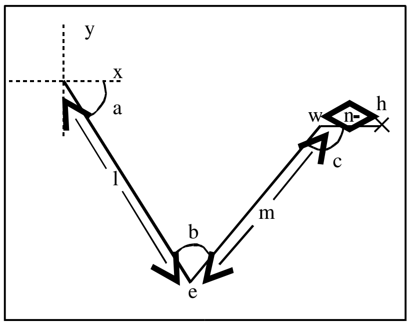
This paper illustrates the steps involved with modelling the software control of a robot arm. A variety of different approaches to modelling the robot arm are considered, the unique difficulties and advantages of using an Empirical Modelling are given for each, and an indication of the relative efficiency of the methods.

As the goal of this paper is to compare the different methods, the concepts and theories behind the methods are only reproduced where necessary to support the work in the paper.

1.1 The underlying model

To compare the different approaches fairly requires a consistent and adaptable background model of the physical attributes of the robot arm.

For simplicity the model is limited to a 2D model of the robot arm, seen from the 'side-on' viewpoint. This limits the arm to five degrees of freedom, simplifying the problem from that faced in real life situations where there are up to ten degrees of freedom with a robot arm. It is assumed that the problems of correctly identifying and normalising the position of the hand and target have already been solved.



1.1.1 Mathematical Model

Figure 1: Values of Interest

In the underlying mathematical model, we are interested in two points, the position of the target and the position of the hand. In this case, the position of the target is directly known, and the position of the hand is calculated from the arm co-ordinates (x and y), the lengths in the arm (l, m and n) and the angles of the joints (a, b and c). The values are shown in Figure 1.

For effective visualisation of the arm, the intermediate points e and w are needed too. The x and y co-ordinates of the points e, w and h are given by the following formulae:

$e_x = x + l * cos(a)$ $e_y = y - l * sin(a)$

$w_x = e_x + m * cos(b)$ $w_y = e_y - m * sin(b)$

$h_x = w_x + n * cos(c)$ $h_y = w_y - n * sin(c)$

The success condition is given by:
$h_x = t_x \ \& \ h_y = t_y$

1.1.2 User Interface

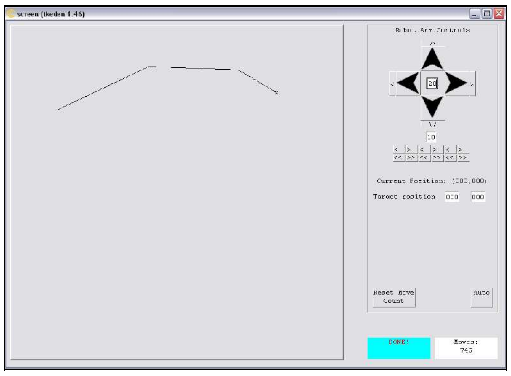


Figure 2: User Interface

Shown above is the user interface that allows all the degrees of freedom to be controlled manually, both in single steps and in larger grouped steps.

It also allows the target position to be set, shows if the arm has reached the target, the number of moves taken so far and allows the number of moves to be reset.

The automatic methods of moving the arm are also selectable from the bottom of the control panel. A visual representation of the arm is given in the main screen on the left hand side of the interface. The target is also represented as a small cross.

1.1.3 Evaluation of performance

To evaluate the various techniques for moving the arm, there must be some method of comparison from one technique to another.

The most simple of these would be to stage a set of experiments with differing tasks required of the arm, and see which of these each technique completed successfully. However, as most of the techniques will be able to solve every task required of them, this is not a very effective method of evaluation.

Instead, the performance will be evaluated by means of a count of the number of individual moves made to reach the goal in each case. Each move consists of rotating a joint by one step, or moving the position of the arm either horizontally or vertically.

To this end, the number of moves taken to get to the current position is shown on the screen, as well as an indication showing if the fingers are over the target.

Also, the opportunity presents itself to ‘cheat’ somewhat by just repositioning the whole arm so the fingers are in the correct position. However, if moving of the arm is disallowed completely then it is possible to move the target outside of the range of the arm, resulting in the arm never ‘finishing’.

Two solutions are possible: either weighting the repositioning of the arm higher than the other movements. Alternatively, the backend will have to trust the arm movement techniques not to move the base position of the arm unless it is necessary. The best solution to this problem can only be chosen once some methods for moving the arm are available.

Overall, the underlying model took longer to implement in EDEN than in a standard object oriented environment. This is more a reflection on the maturity and design of the current tools than the credibility of EM techniques.

The individual differences between the three scripting languages required to make the model work were often a source of frustration and error. The repetitive nature of adding buttons to the toolbar was reminiscent of manual calculation that computers were first designed to replace.

1.1.4 Extensions to the model

If there was more time available the model could be expanded to have a graph of the distance from the target, allowing a more in depth analysis to be made. Properties such as minimising rotations of a particular joint or spreading the movement out due to overheating issues or similar physical constraints.

The model could also be extended to provide ‘no-go’ areas that could represent blocking objects or obstacles that the arm could face in a real situation or to deal with any mechanical limitations of the joints in the arm.

The model could be extended in to three dimensions quite simply by using the Sasami notation. The additional problems of collision detection and accurate modelling would then have to be solved.

If the model were extended in to 3D, as well as the other improvements given above, it would be of a standard that could be used to solve simple problems faced by actual robot arms.

Indeed, if the model was deemed useful for real world applications then a real robot arm could be attached to a computer, and an interface to the *EDEN* environment designed, to allow it to be moved directly. This would also involve input to the system as feedback to the model.

2 Automated movement

The following section discusses various different methods that could be used to move the arm, how they could be implemented in the model, and the difficulties associated with each.

2.1 Hill-Climbing

2.1.1 Overview

One of the simplest AI techniques that can be applied to this problem is that of 'hill climbing', also known as greedy local search. Due to time constraints, this is the only method actually implemented in the model.

The idea behind this method is to simply look at the current state, and choose the action that makes the biggest improvement. If there are several moves that would make the same improvement then choose one at random. Figure 3 is the 'elevation map' of a typical situation in the implemented model. The target is shown as a cross, with 'higher' areas shown in a darker colour, denoting a point closer to the target.

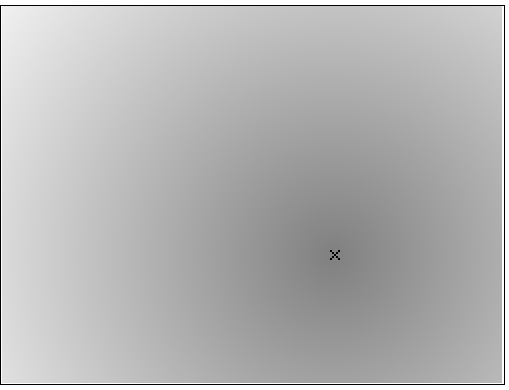


Figure 3: Simple elevation map

2.1.2 Implementation

The hill climbing model requires two considerations to operate. The first is a representation of the current state. This is stored implicitly in the state of the model, namely the in variables *arm_x, arm_y,* etc.

The second is some way of evaluating every possible move available. The easiest way of achieving this in EDEN is the introduction of a set of decision variables that give the position of the arm in each of these adjacent states.

For these adjacent states, the intermediate points of the arm are not important, so the calculation can be designated to a function.

The required behaviour is achieved by simply calculating the distance between the points and the target using Pythagoras:

$$D=\sqrt{(P_x-T_x)^2+(P_y-T_y)^2}$$

All that remains is to select which of these has the smallest distance D, and to make the move associated with that variable.

2.1.3 Evaluation

Given simplicity of the model, you would expect that the performance of the arm would be of a low standard when using this method of moving the arm. In most situations the arm performs amicably, the situations where it performs poorly are ones where a smaller initial advantage in distance would lead to a movement with less moves overall. This is consistent with the greedy nature of the method.

The dependency method of programming was of great advantage with the implementation of this movement method. The update of the decision variables is handled completely automatically by the system, removing the need to write triggered functions to update the large number of values that the variables depend on.

2.1.4 Dealing with Extensions to the Model

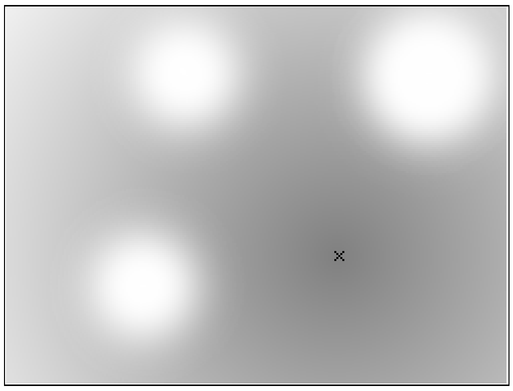


Figure 4: Advanced elevation map

Due to the simple nature of the method, it is easy to adapt for the various extensions that are possible in the model.

For example, to deal with blocking objects involves a slight adjustment to the distance function to take account of the objects. An example of this type of elevation map is given in figure 4. The lighter areas represent the ‘pits’ in the map made by the blocking objects.

If the model were extended in to three dimensions, the method would have to consider two extra decision variables for each extra degree of freedom that was given to the arm.

2.2 A* Informed best-first search

To overcome the limitations of the hill-climbing method, it is necessary to consider a more advanced form of AI agent. The most important difference in this case is that the route is calculated beforehand. This allows the most optimal movement to be decided before making any move that might not be part of that movement.

The A* search technique is the best of the informed, search based AI techniques. A search tree is built of all the possible states that could be reached from the previous state, with the initial position of the arm forming the root of the tree. Each node in the tree is given a ‘cost value’ which is made up of the cost of getting to that state form the initial state, and an estimate of how far that state is from the goal. The tree is expanded by looking at the successor states to the node with the smallest cost value.

The advantages of this method come through the sacrifice of the adaptability in the hill climbing method. If the target changes position then the whole route needs to be recalculated. This quickly becomes computationally expensive, and rules out any possibility of ‘catching’ a moving target.

The difficulties in implementing this method would be storing and calculating the search tree. The final search tree could have tens of thousands of nodes, any of which could be needed at any time during the route building process. The globally accessible, loosely typed nature of EDEN makes storing and accessing large amounts of structured data more onerous than in a traditional programming language.

2.3 Human Emulation

Another approach to the movement of the arm is to consider the natural method that is used by the human arm.

Figure 5: Human method

From empirical observations, the method is split in to two major parts. The first is aligning the hand towards the object, while simultaneously moving the wrist to lie on a line between the shoulder and the target. The second is the extension of the arm until the fingers are in the correct position, including moving the shoulder forwards if required. Figure 5 shows the arm after stage one of this.

The implementation of this method poses an interesting problem. The first part of the task is almost equal in difficultly to the whole task, knowing which way to rotate the arm and knowing when it is as close to the line as it can be involves similar decisions as the whole task.

Even when the arm is aligned correctly, extending it in a straight line is not simple, involving a combination of trigonometric and the ratio of the component lengths in the arm.

2.4 Scheduling approach

As discussed in Figueroa (1997) when a real time programming approach is taken to the control of the robot arm, the solution is equivalent to that of a “general real time scheduler”.

Bearing close relation to parts operating system design, there are several different scheduling algorithms, including static, priority driven and dynamic (Ramamritham and Stankovic, 1994).

Although not specifically in the area of real time scheduling, a body of work already exists exploring an empirical approach to scheduling, resulting in “The Temposcope”, a tool designed to assist a naive user in solving the tutorial scheduling problem.

It would be interesting to see if dependency based modelling could be used to enhance real time scheduling, either through adapting the previous scheduling work or through a completely new modelling exercise.

The amount of effort involved to develop such a system for the control of the robot arm through EM would seem to far outweigh the benefits, except if it were done with reference to the exploration of dependency based scheduling, as detailed above.

2.5 Other considerations

Much of the current work in robot arms seems to be the control of the arm’s movements though the use of neural driven interfaces, both in primates

(Nicolelis, 2003) and in human subjects (Warwick, 2002). The roll of EM in this area seems unclear, as the system's response to the neural activity in nerves is greatly complex, and modelling this response and other factors are out from scope of current tools.

3 Object enriched EM

The following section discusses how the combination of object orientation and empirical modelling could be achieved, what benefits it could bring, and the difficulties involved combining the two techniques.

The motivation for changing the current tools is threefold.

Firstly, the current set of tools is inconsistent. While it is advisable to split the underlying system from the extensions that provide e.g. visual layout tools, there must be some notion of consistency between them. Otherwise, the effort of becoming proficient in a new language is tripled.

For example, in the current tools the DoNaLD and Sasami notations do not require semicolons at the end of statements, but SCOUT and EDEN do. To access an EDEN variable in SCOUT, it is necessary to re-declare it in SCOUT. To access an EDEN variable in DoNaLD you append ! to the variable name. These differences make scripting in the notations more difficult than it needs be.

Secondly, the lack of some level of collective identity for variables leads to so called 'magic values', or in this case 'magic variables'. When the textbox x's content is changed, it sets the variable x_TEXT_1 to the content that was changed. Not only does this pollute the global namespace with unnecessary variables, but also it is less natural than say x.text, as would be seen in an object-oriented approach. This is also true for setting the appearance of the DoNaLD shape x by setting the EDEN variable A_x.

Finally, there is the issue of familiarity. For most students, the programming language they are most familiar with is Java, a strongly typed object based language. The transition to EDEN, a weakly typed scripting langue with no object system can be a step in to the unknown.

Clearly, there are some fundamental concepts to EM and object orientation that cannot be easily combined. The idea of encapsulation in good object oriented practice sits at odds with the freeness and openness that is at the core of EM. When such issues arise, it should normally be the EM principle that takes precedence over the object oriented one. The goal of object enrichment is not to undermine the foundations of EM, but to supplement and support them.

There are, however, exceptions to this rule. The introduction of strong typing would seem to reduce the options available to us, removing some freedom that was available to us in EDEN. However, the underlying code is unlikely to be able to deal with the unexpected data in a meaningful way. For instance, setting what was a numerical value of speed to the string "fast" is likely to produce undefined results, a situation that is rarely profitable, even in EM.

The introduction of a more structured method of EM modelling would also enable greater reuse of code, something that has gone unrealised in the current tool set. Considering other models during EM design work is more often than not just a matter of inspiration, rather than that of serious reuse. A library of commonly used agents could be introduced, speeding the construction of more complex models. For instance, there could be an agent (which in turn could control many other sub-agents) that sorted a given list of strings in to alphabetical order, or an agent that stored a representation of a binary tree. These agent-object hybrids would not hide the internal representation of themselves like traditional object, merely 'guide' other agents to their correct observables, in the correct order.

In conclusion, while the difficulties may appear minor, added together they can turn frustrating very quickly. The object-enriched system would overcome most of the downfalls and bring other advantages, but many could also be solved by ‘revamping’ the existing tools to be more consistent.

References

Ramamritham K., Stankovic J.A -
Scheduling Algorithms and Operating Systems Support for Real-Time Systems
-http://citeseer.ist.psu.edu/
ramamritham94scheduling.html - (1994)

Figueroa, M. A. - The control of a toy robot ARM: a real time programming experience -
http://portal.acm.org/citation.cfm?id=31726.31791 - (1997)

Nicolelis M. - Monkeys Consciously Control a Robot Arm Using Only Brain Signals
http://dukemednews.org/news/article.php?id=7100 - (2003)

Warwick K. - Project Cyborg 2.0:
The next step towards true Cyborgs? -
http://www.rdg.ac.uk/KevinWarwick/html/project_cyborg_2_0.html - (2002)