Fault Tolerant Control using Cartesian Genetic Programming

Yoshikazu Hirayama
University of York
York, UK
YO10 5DD
yh120@ohm.york.ac.uk

Tim Clarke
University of York
York, UK
YO10 5DD
tim@ohm.york.ac.uk

Julian Francis Miller
University of York
York, UK
YO10 5DD
jfm@ohm.york.ac.uk

ABSTRACT

The paper focuses on the evolution of algorithms for control of a machine in the presence of sensor faults, using Cartesian Genetic Programming. The key challenges in creating training sets and a fitness function that encourage a general solution are discussed. The evolved algorithms are analysed and discussed. It was found that highly novel, mathematically elegant and hitherto unknown solutions were found.

Categories and Subject Descriptors

I.2.8 [**Artificial Intelligence**]: Problem Solving, Control Methods, and Search; I.2.9 [**Artificial Intelligence**]: Robotics—*Sensors*; F.2.2 [**Analysis of Algorithms and Problem Complexity**]: Nonnumerical Algorithms and Problems

General Terms

Algorithms, Reliability

Keywords

cartesian genetic programming, evolutionary algorithms, sensor fault tolerance, control

1. INTRODUCTION

Sensors on a physical system provide crucial information: the status of the system and/or the environment in which it is situated. However, they are fallible. A faulty sensor signal may lead to wrong control system behaviour and bring about an undesirable situation for the system. A reason [14] for little literature on sensor fault tolerant control is the critical role of measured variables in a controlled system requiring high reliability - often achieved through the use of direct (hardware) redundancy; Multiple sensors are utilised and majority voting used for the selection of healthy sensors. Also, a faulty sensor can be replaced, physically, by spare sensors, if they exist. In combination with direct redundancy, or on its own, analytical redundancy can also be used as part of the fault tolerant control system (FTCS) design. Analytical redundancy is provided by a model of the system variable of concern, and produces an estimated value in lieu of the faulty sensor. For example, in a control system with a sliding mode observer unit [5][3], faulty sensors are replaced by their estimations.

In this paper, the authors offer a novel alternative method for tolerating sensor failure in a control system, where multiple sensors are required for measurement. It is done without reconfiguring the control laws, or without having to estimate the correct values from the faulty sensor values. This is achieved by focusing on generating the correct inputs to the controller which are normally calculated based on a full set of working sensor values. A type of evolutionary algorithm called *Cartesian Genetic Programming* (CGP) [11] was used to evolve solutions which can generate the appropriate controller input values but using only the remaining, working sensors. Using only the remaining sensors means that spare sensors are not necessary, reducing the cost or inconvenience in the system hardware design. However, they could be used as an adjunct to direct redundancy for higher reliability when all else has failed.

We built a generic demonstrator, *Shaky Hand*, to test the evolved solutions in practice. The demonstrator utilises multiple sensors, the data from which are integrated to give information about the status of the system. This integration process from data to information, means that Shaky Hand represents a natural model of a real industrial machine. Shaky Hand is suitable for testing sensor fault tolerant and data fusion techniques due to its multiple sensor environment and also because the quality of sensor data affects the quality of the output information. One can compensate or enhance the sensor data using these techniques to improve the quality of the output information.

CGP has demonstrated its effectiveness in learning Boolean functions over conventional GP [8] and has been applied in variety of applications. These include digital circuit designs [10], image filter and its implementation in FPGA [7][13], artificial life [6], bio-inspired developmental models [9], and evolutionary art [1][4]. However, the use of CGP in sensor fault tolerant control application has not been explored before. The CGP based sensor fault tolerant control is also novel in the field of the sensor fault tolerance. Since the outcome of CGP can be analysed, the system reliability can be enhanced, and this can be considered practical. These are the motivations for applying CGP for the sensor fault tolerant application. The work demonstrates that sensor fault control applications can benefit from the use of CGP.

GECCO'08, July 12–16, 2008, Atlanta, Georgia, USA.

1.1 Structure of the Paper

The Shaky Hand system is briefly introduced first including its description of the control scheme and the plate sensors. Secondly, we discuss the method used to allow CGP to evolve generic solutions which have fault tolerant ability. Thirdly, the evolved solutions are shown and analysed to show why they work well. The last section concludes the work.

2. THE SHAKY HAND SYSTEM

In this section, the novel laboratory demonstrator which we named *Shaky Hand* is introduced.

2.1 The Shaky Hand Physical System

The Shaky Hand game was modelled on a village fête game. In the original, as shown in Fig.1, the aim is to guide, by hand, a wire loop along a meandering wire track from one end to another, without touching the loop to the wire. When the loop touches the wire an electrical circuit is made and an alarm is set off.

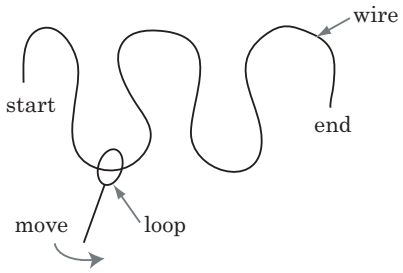

Figure 1: Outline of the Shaky Hand game

Shaky Hand follows this model. However, the loop is guided by a flat bed plotter arrangement with x and y translational drive motors as shown in Fig.2.

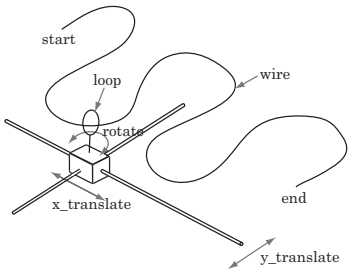

Figure 2: Mechanising the Shaky Hand game

These are rotary DC motors, driving their load via leadscrews. The loop can be rotated by a third DC motor to keep the plane of the loop perpendicular to the wire. Four inductor coils on a plate just below the wire allow the proximity and orientation of the wire to be sensed (Fig.3).

The wire carries an alternating current of appropriate magnitude and frequency. The magnitude of the induced emf in each coil is inversely proportional to wire proximity. The output voltages from the four coils are then amplified and converted into DC signals and presented to a PC based analogue data acquisition card. To make an interesting scenario we define a set of game 'rules'. The loop must be guided from one end of the wire to the other and should never touch the wire. Loop size is defined by the sensor positions, so the wire should never touch the sensors. We define this as a catastrophic failure. We also add time constraints, otherwise the movement of the loop may stop as the system decides what action to take next. So we define another catastrophic failure condition: the speed of the loop along the wire direction shall be kept at a defined constant level which is non-zero.

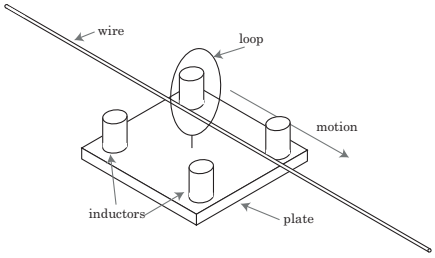

Figure 3: The Shaky Hand game pickup plate

2.2 Plate Sensors

The work described in this section focuses on the four coil sensors mounted on the Shaky Hand plate. The four sensor outputs are used directly to obtain a lateral offset error voltage, V_α, and an angle offset error voltage, V_ϕ, caused by the misalignment between the centre of the plate and the track. These voltage errors are used to control plate movement. The wire is assumed to be locally straight. Fig.4 displays the sensor arrangement on the plate. The wire track passes between the top sensors (sensors A and B) and the bottom sensors (sensors C and D).

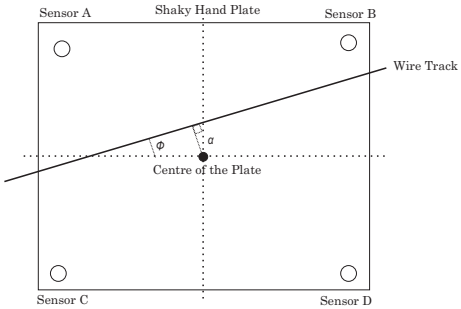

Figure 4: Four sensors on the Shaky Hand plate and the offset terms

The lateral offset, α is obtained in terms of the voltage V_α:

$$V_\alpha = (V_A + V_B) - (V_C + V_D) \quad (1)$$

and the angle offset, ϕ is obtained in terms of the voltage V_ϕ:

$$V_\phi = (V_A + V_D) - (V_B + V_C) \quad (2)$$

The terms V_A, V_B, V_C and V_D used in Equations 1 and 2 are the output voltages from the sensors A, B, C and D on the plate respectively. The offset voltages are the inputs to the controllers which drive the appropriate motors to compensate for the offsets.

3. SENSOR FAULT TOLERANCE BY CGP

A novel evolutionary programming approach to generating offset error voltages in the presence of sensor coil failure is now presented. This includes the exploitation of training data sets that avoid over-fitting, which ensures that the resultant algorithmic solutions are generic and therefore will work with any wire track shape.

Equations 1 and 2 assume that all the sensors function correctly and output appropriate signals. However, they become invalid when one or more sensor fails. Now, CGP is used to evolve the offset error sensing solutions which utilise less than four sensor outputs yet still provide a reasonably accurate estimation of the two offset errors. This is depicted in Fig.5.

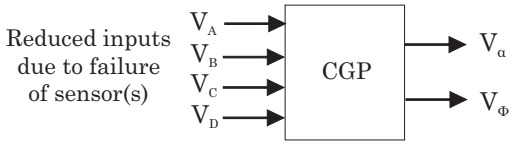

Figure 5: CGP for Shaky Hand

Assuming, for now, that Shaky Hand is able to detect the faulty sensor(s) and subsequently select appropriate offset error sensing solutions according to the sensor fault condition, Shaky Hand would then be able to continue to operate, perhaps with degraded performance. The coil sensor outputs are normally non-zero, positive values so we reasonably assume that, under failure conditions, the sensor outputs are reset to zero.

3.1 CGP and its application to the Sensor Failure Problem

Cartesian Genetic Programming, which developed from the work of Miller and Thomson [12], represents programs by directed acyclic graphs. CGP use a rectangular grid of rows and columns of computational nodes. With nodes in the same column not be allowed to connect to each other. It also uses a connectivity parameter called *levels-back* which determines how many columns on the left a node may connect to. The genotype is a fixed length list of integers, which encode the function of nodes and the connections of the directed graph. It also has a number of output genes that encode connection points in the graph where program outputs are taken from. When the number of rows is chosen to be one and *levels-back* is set to the number of columns, the genotype encodes an arbitrary acyclic graph. This means nodes can take their inputs from either the output of a previous node or from a program input (terminal). We have chosen this for the work in this paper. The number of inputs that a node has is dictated by the number of inputs that are required by the function it represents. The phenotype is obtained by following the connected nodes from the program outputs to the inputs. It is important to note that in this process, some node outputs may not be used so that their genes have no influence on the final decoded program. Such non-coding genes have no effect on genotype fitness.

In this paper an evolutionary strategy [2] has been used of the form $1 + \lambda$, with λ set to 4, i.e. one parent with 4 offspring (population size 5). This is typical of many implementations of CGP. In this evolutionary algorithm the parent, or elite genotype, is preserved unaltered, whilst the offspring are generated by mutation of the parent. While best chromosome is always promoted to the next generation, if two or more chromosomes achieve the highest fitness then the newest (genetically) is always chosen [10].

For Shaky Hand, the inputs to the CGP are the four plate sensor signals, V_A, V_B, V_C and V_D and the outputs are the two offset error voltages, V_α and V_ϕ. Mutation rate is defined as the percentage of genes that are mutated. This was chosen to be 1%. The number of generations was limited to 50,000. One hundred, two input, single output functional blocks of the type depicted in Fig.6 were chosen. This allowed a rich variety of multi-sensor inputs/single offset voltage output algorithms to be evolved. The number of functional blocks determines the length of the integer representation. Each block contains three genes representing two incoming node connections and one operator type, combining this with the output. There are 302 genes in total. This genotype is sufficient to produce relatively complex solutions.

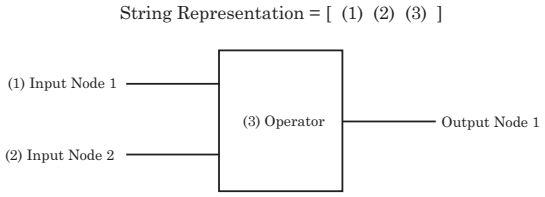

Figure 6: A single CGP functional block representation

One hundred input sets are used and the fitness of the evolved solutions are evaluated per generation using Equations 3 to 6.

$$J_{V_\alpha} = \left| \frac{V_{\alpha_{ideal}} - V_{\alpha_{evolved}}}{V_{\alpha_{ideal}}} \right| \quad (3)$$

$$J_{V_\phi} = \left| \frac{V_{\phi_{ideal}} - V_{\phi_{evolved}}}{V_{\phi_{ideal}}} \right| \quad (4)$$

If ($V_{\alpha_{ideal}} = 0$) or ($V_{\phi_{ideal}} = 0$), then the equations are modified to:

$$J_{V_\alpha} = \left| V_{\alpha_{ideal}} - V_{\alpha_{evolved}} \right| \quad (5)$$

$$J_{V_\phi} = \left| V_{\phi_{ideal}} - V_{\phi_{evolved}} \right| \quad (6)$$

A normalised cost function was used because of its property of forcing the evolution of good solution algorithms when the actual offsets, α and ϕ, are very small. Since the loop closure of the control system will tend to drive the offsets towards zero (good wire tracking), it is important that the sensor failure algorithms should work best under tight tracking. Fig.7 illustrates how the cost is evaluated using the evolved and the original algorithm outputs. The output sets from the original algorithms are defined as ideal output sets in this figure.

To simulate a sensor failure, one member of the input sets is forced to be zero. It is reasonably assumed that Shaky Hand has a fault detection system such that when a failure is detected the signals from the failed sensors are nulled. Using

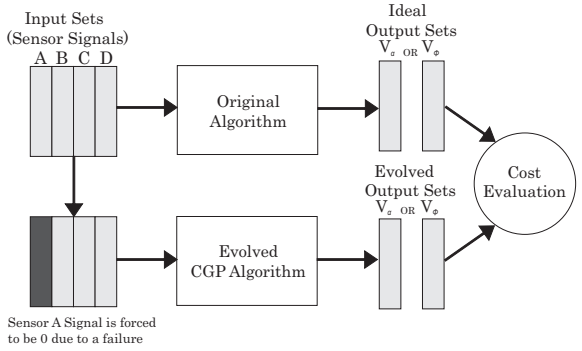

Figure 7: Comparing the ideal and the evolved solutions

these input sets, solutions are evolved. For each generation, the output sets from all the solutions in the population are compared with the ideal output sets to determine the best one. The generation of the new population ends when a stopping criterion set by the user is met. In theory, the best possible cost value is 0, which would mean the successful evolution of identical output sets to the fault-free originals. However, the sensor failures are expected to cause deteriorations and 0% error may not be achieved. Therefore the criterion for the convergence was initially set to 0.01. If the cost is less than 0.01 the solution is considered to have the equivalent response to the original algorithm. i.e. the outputs produced by the current best solution in the population for 100 incoming data have less than 1% error. A 1% error in measurement would have negligible effect on the tracking along the wire by the Shaky Hand plate. The negative feedback control would act on this error as if it was a disturbance. The disturbance rejection properties of negative feedback are well known and documented in all good elementary control engineering text books.

A wide range of operators is provided including primitive and conditional operators so the evolutions would have variety of choice of the operators. They are described in Table 1. X_1 indicates the input node 1 of one functional block, X_2 is the input node 2. O is the output node of a functional block. For the evaluation of the output of the solution, exception handling is incorporated into some operators, such as a divide by zero, as protected functions. Conditional operators (operators 10 and 11) allow more complex solutions to be evolved, providing solution choices according to the values assigned to the input nodes. Depending on the situations, a solution can therefore have totally different values and can better adapt to dynamic changes in the environment.

3.2 Genericity

For the Shaky Hand CGP, the solutions are not evolved through the physical environment but a virtual one. The environment for the Shaky Hand case is the wire track shape provided by the training data sets. This environment must be sufficiently open to avoid over-fitting. A closed environment would over-specify the system, shaping its behaviour to that particular environment only, creating solutions that work on a particular wire track only. This case had to be avoided, so a virtual environment was designed to achieve sufficiently rich interactions between the system and itself. Methods used in the Shaky Hand CGP to achieve such an

Table 1: Operators used in the Shaky Hand CGP

Operator indices	Operator types	Protected functions
1	Addition (O=X_1+X_2)	
2	Subtraction (O=X_1-X_2)	
3	Multiplication (O=$X_1 \times X_2$)	
4	Division (O=$\frac{X_1}{X_2}$)	if X_2=0, O=0
5	Square (O=X_1^2)	
6	Square root (O=$\sqrt{\lvert X_1 \rvert}$)	Use absolute value
7	Reciprocal (O=$\frac{1}{X_1}$)	if X_1=0, O=0
8	Natural log (O=lnX_1)	if $X_1 \leq 0$, O=0
9	$\log_{10}$ (O=$\log_{10}X_1$)	if $X_1 \leq 0$, O=0
10	Max (O=max(X_1,X_2))	
11	Min (O=min(X_1,X_2))	
12	Absolute value (O=$\lvert X_1 \rvert$)	
13	Sine (O=sinX_1)	
14	Cosine (O=cosX_1)	
15	Tangent (O=tanX_1)	if X_1=(n+$\frac{1}{2}\pi$) n=(0,l,2...), O=0
16	Power (O=$X_1^{X_2}$)	if X_1=0, O=0
17	Sign change (O=-X_1)	

open and rich environment are discussed below. There are three major aspects; the special training set, sliding windows, and the use of multiple virtual tracks.

3.2.1 *Training Set*

Input data sets to the program are the signal values from sensors A, B, C and D on the plate. The sensor values should be consistent with the correct operation of the Shaky Hand system. The signals must be continuous and without repetition, representing realistic input sets that do not present a closed environment. In order to satisfy these criteria, a model relating α and ϕ with the physical dimension of the plate was created (Fig.8).

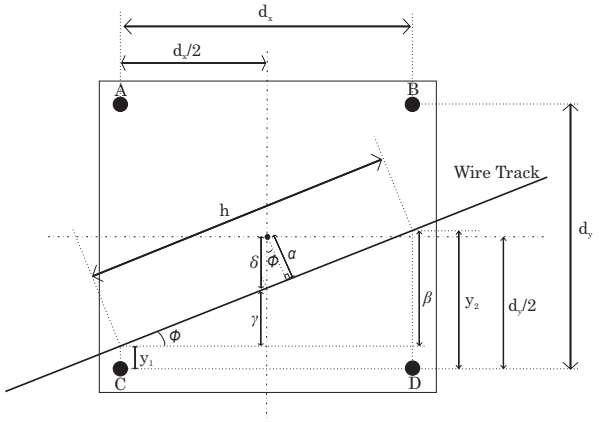

Figure 8: Plate model to create CGP input sets

By altering the wire position at the edge of the plate continuously, through y_1 and y_2, and without repetition, α and ϕ are, in turn, altered continuously and without repetition. The sensor signals are then generated as required. From

Fig.8,

$$\tan\phi = \frac{\beta}{d_x} \tag{7}$$

Since

$$\beta = y_2 - y_1 \tag{8}$$

The angle offset, ϕ, is

$$\phi = \arctan(\frac{y_2 - y_1}{d_x}) \tag{9}$$

By scaling,

$$\gamma = \frac{\beta}{2} = \frac{y_2 - y_1}{2} \tag{10}$$

δ is defined as

$$\delta = \frac{d_y}{2} - \gamma - y_1 \tag{11}$$

Therefore,

$$\delta = \frac{d_y - y_2 - y_1}{2} \tag{12}$$

and also,

$$\alpha = \delta\cos\phi \tag{13}$$

α in Fig.8 has a negative sign by convention and from Equations 12 and 13,

$$\alpha = -\frac{d_y - y_2 - y_1}{2}\cos\phi \tag{14}$$

Equation 14 can be further simplified.

$$h = \sqrt{d_x^2 + \beta^2} = \sqrt{d_x^2 + (y_2 - y_1)^2} \tag{15}$$

$$\cos\phi = \frac{d_x}{h} = \frac{d_x}{\sqrt{d_x^2 + (y_2 - y_1)^2}} \tag{16}$$

Substituting Equation 16 into 14 gives

$$\alpha = -\frac{d_y - y_2 - y_1}{2}\frac{d_x}{\sqrt{d_x^2 + (y_2 - y_1)^2}} \tag{17}$$

So,

$$\alpha = -\frac{d_x(d_y - y_2 - y_1)}{2\sqrt{d_x^2 + (y_2 - y_1)^2}} \tag{18}$$

The continuous and non-repeating y_1 and y_2 data are obtained using the Matlab *polyfit* function. This generates a polynomial of predefined order that fits data points provided by a user. In this case, two 9th order polynomials were generated to express y_1 and y_2 variations as shown in Fig.9. Data points were chosen so that both polynomials vary within the range constrained by the physical size of the plate. The order of the polynomials was chosen to provide reasonably smooth curves. The term, 'sliding window', on this figure is explained later. Using the Matlab *polyval* function, 5000 data points each were extracted from the y_1 and y_2 polynomials. They become the *data bank* for the input sets of the Shaky Hand CGP. Equations 9 and 18 are then used to convert the wire positions provided by the data bank into the offset errors. The offset errors are then converted into the sensor voltages. Continuous sequences of sensor voltages become the input sets for the Shaky Hand CGP.

3.2.2 Sliding Windows

The Shaky Hand CGP takes in 100 consecutive input data points per generation from each of the y_1 and y_2 data banks. The program selects the starting data points at random for both y_1 and y_2 data displayed as A_0 and B_0 in Fig.9. 100 consecutive data points from the starting data points, enclosed by the Sliding Window are selected and then converted into the sensor signals V_A, V_B, V_C, and V_D as shown previously using α and ϕ. The combination of randomly and independently selecting the start points A_0 and B_0 over these two long data sets means that the training data sets are realistic yet, to all intents and purposes, unrepeated over very many experiments. The Sliding Window on y_1 data bank is shifted by 10 data points to the right every 1000 generations, and the window on y_2 data bank is fixed. In other words, the input sets are kept the same for 1000 generations and are then modified over 10% of their range. Using the modified input sets, the solution is evolved again. This gradual rather than an abrupt change in input sets, helps the evolution to migrate towards generic solutions. The window on y_2 data bank is fixed, yet, the variation in input sets is still enormous.

3.2.3 Multiple Virtual Tracks

In a further effort to ensure the genericity of CGP solutions, training sets were further modified. A generic solution means it works on any given input set. Therefore, two different input sets were selected and applied to the evolutionary processes. i.e. the cost of a solution is evaluated using two totally different virtual wire tracks. The two starting data points are chosen from each of the y_1 and y_2 data bank as shown in Fig.9.

3.2.4 Test Set

The stochastic nature of CGP meant that the obtained solutions could be different at every run. The requirement was for generic solutions, so a genericity test was devised as follows; The test input sets were characterised from y_1 and y_2 data in the data bank described in Section 3.2.1. Each bank consisted of 5000 data points and, in this case, the order of y_1 data was reversed. All of the reversed y_1 and non-reversed y_2 data were used as the test sets. Because of the reversing, the test sets would look different from the training sets. The 5000 test sets were applied to each solution and the mean costs were analysed and compared with each other. A solution with the best mean cost out of 60 obtained solutions was selected as the best solution.

3.3 Evolved Solutions

CGP was used to evolve solutions for one sensor failure cases and the obtained solutions are shown here. The symmetry of the plate means that if one good solution is obtained then that solution can be modified to fit other equiv-

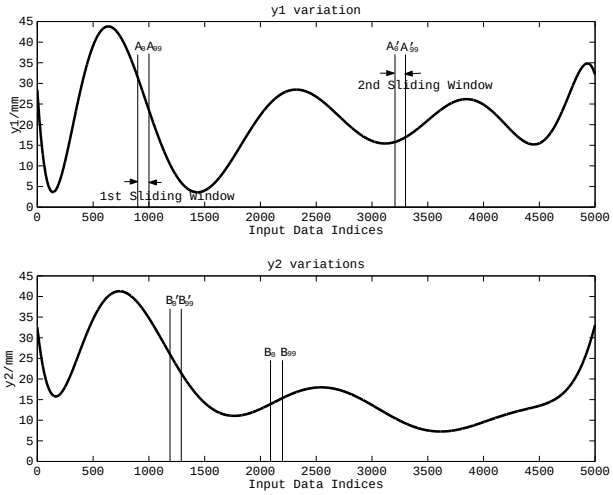

Figure 9: y_1 and y_2 data variations with two sliding windows each

alent, complementary sensor failure cases.

1. Sensor A Failure

$$V_\alpha = \ln((\frac{V_B}{V_C})^2) \quad (19)$$

$$V_\phi = \frac{V_D}{V_C} - \frac{V_C}{V_D} \quad (20)$$

2. Sensor B Failure

$$V_\alpha = \ln((\frac{V_A}{V_D})^2) \quad (21)$$

$$V_\phi = \frac{V_D}{V_C} - \frac{V_C}{V_D} \quad (22)$$

3. Sensor C Failure

$$V_\alpha = \ln((\frac{V_A}{V_D})^2) \quad (23)$$

$$V_\phi = \frac{V_A}{V_B} - \frac{V_B}{V_A} \quad (24)$$

4. Sensor D Failure

$$V_\alpha = \ln((\frac{V_B}{V_C})^2) \quad (25)$$

$$V_\phi = \frac{V_A}{V_B} - \frac{V_B}{V_A} \quad (26)$$

Each V_α and V_ϕ solutions utilise only two sensor signals rather than three. However, all the three sensor signals are required to obtain full information for the position of the center of the plate relative to the wire track.

4. ANALYSIS

The first analysis evaluates the mean fitness of each solutions over 5000 test input sets to show their genericity. The second analysis describes why the solutions work well.

4.1 Fitness Measurement

Using the validation data sets, the mean costs, J_{V_α} and J_{V_ϕ}, over the test input set (5000 input sets) for the best evolved V_α and V_ϕ solutions and their standard deviations are summarised in Table 2.

Table 2: Summary of the cost and the standard deviations of the evolved solutions

Failure case	J_{V_α}(STD)	J_{V_ϕ}(STD)
Sensor A failure	0.312(0.222)	0.206(0.215)
Sensor B failure	0.241(0.256)	0.206(0.215)
Sensor C failure	0.241(0.256)	0.299(0.297)
Sensor D failure	0.312(0.222)	0.299(0.297)

The one sensor failure cases have mean costs of less than 0.5. This indicates that for every input set used, an error of less than 50% is made on the estimation of the wire position on the plate relative to the sensor positions. Looking into the physical size of the plate (45mm by 45mm), if the cost is relatively low, for example less than 0.5, the centre of the plate will be close to the wire track, e.g. if α is 5mm, then the error would be approximately 2.5mm. An error-driven control algorithm will drive the plate to reduce this error, provided that the sense of the error is in the correct direction. So, as long as the motors move correctly and α and ϕ are small, then Shaky Hand should operate correctly but with degraded tracking performance. If the offsets are large and/or of the wrong sense, combined with high cost then there would be a serious problem.

4.2 Why Do the Algorithms Work So Well?

The solutions for the one sensor failure cases (Equations 19 to 26), are elegant. Analytical reasonings behind these solutions are discussed here.

4.2.1 Analysis of V_α Case

The V_α solutions utilise diagonal sensor pair outputs. Let us look into Equation 19, where sensor A (or D) has failed. If the wire track is situated in the centre of the plate then V_B and V_C are equal, giving $V_\alpha = 0$. If the wire is closer to sensor B than to sensor C, then V_B is larger than V_C. So, V_B/V_C is greater than 1. Taking natural logarithmic of the value provides positive value. If V_C is larger than V_B, then, the solution provides a negative value as V_B/V_C is a fraction. So, the natural logarithmic term gives the correct polarity of α. A square term in the equation gives an amplification effect, providing greater penalty in the presence of α, which drives the plate back into the correct position quickly through control action.

During the evolution of the solution, the program identified the natural logarithmic function rather than the log_{10} function which could also provide the correct polarity. However, the natural logarithmic function generates a stronger penalty effect in the presence of α.

4.2.2 Analysis of V_ϕ Case

Let us look into Equation 20 (Sensor A or B failure case) for the V_ϕ solution. It uses two adjacent sensors to evaluate the output. V_D is normalised to V_C by the term V_D/V_C and V_C is normalised to V_D by the term V_C/V_D. The difference is taken as V_ϕ. It gives correct polarity at all times. So, how does the evolved solution differ from the original solution? Fig.10 illustrates the plate configuration.

Initially, we assume the presence of ϕ, with α=0. So, $\triangle C = \triangle D$. The voltage induced in the coil is inversely proportional to its proximity to the wire track and has a

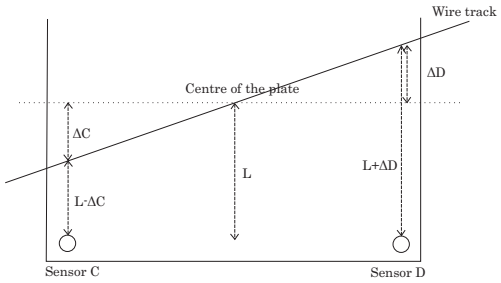

Figure 10: Plate configuration for the V_ϕ solution analysis

constant proportionality which we denote K. So,

$$V_C = \frac{K}{L - \triangle C} = \frac{K}{L - \triangle D} \tag{27}$$

$$V_D = \frac{K}{L + \triangle D} \tag{28}$$

The evolved solution (Equation 20) can be represented as

$$V_\phi = \frac{K}{L + \triangle D} \frac{L - \triangle D}{K} - \frac{K}{L - \triangle D} \frac{L + \triangle D}{K} \tag{29}$$

$$= -\frac{4L\triangle D}{L^2 - \triangle D^2} \tag{30}$$

Let us now add an offset α, so, L→L+α,

$$V_\phi = -\frac{4(L + \alpha)\triangle D}{(L + \alpha)^2 - \triangle D^2} \tag{31}$$

The evolved solution (Equation 31) is now compared with the original solution:

$$V_\phi = (V_A + V_D) - (V_B + V_C) \tag{32}$$

In Fig.10, V_A and V_B are the same as V_D and V_C respectively. Therefore, using substitution and simplification, the original solution (Equation 32) becomes

$$V_\phi = -\frac{4K\triangle D}{L^2 - \triangle D^2} \tag{33}$$

Equation 33 is very similar to Equation 30 which is the evolved solution. In fact, fortuitously, K and L do have the same value. Therefore these solutions are exactly the same when there is no lateral offset. When α is present, Equation 33 becomes

$$V_\phi = -\frac{4K\triangle D}{(L + \alpha)^2 - \triangle D^2} \tag{34}$$

So the term (L+α) in Equation 31 no longer matches K in Equation 34, which does not change in the presence of α. The graphs plotted based on Equations 34 and 31 are shown in Figs. 11 and 12 respectively. The graphs plot the curves of V_ϕ over a range of ϕ under the influence of α. The dashed line in each plot represent the V_ϕ values when α=0.

Graphs for the original and the evolved solutions show similar pattern. However, the magnitude of V_ϕ is different. The magnitude of V_ϕ from the evolved solution is smaller

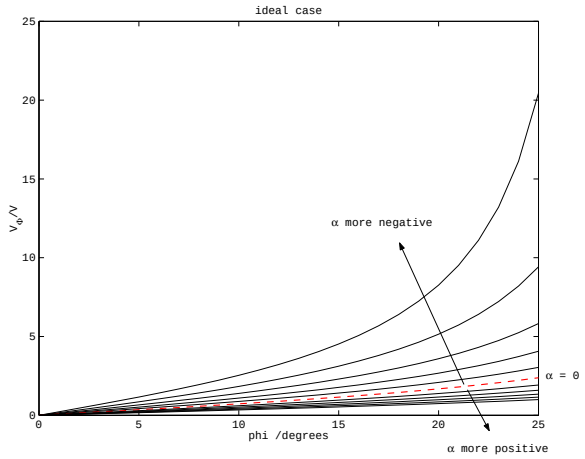

Figure 11: V_ϕ against ϕ with different α values using the original solution

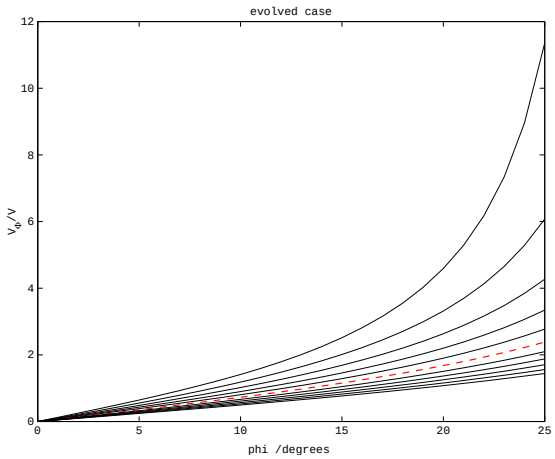

Figure 12: V_ϕ against ϕ with different α values using the evolved solution

when α is negative, and is larger when α is positive. The fitness is higher (i.e. the difference between the original and the evolved solution becomes less) when α becomes smaller. The evolved solution can compute exactly the same V_ϕ as the original solution when α=0. Therefore, as long as α is kept small, V_ϕ from the evolved solution is kept close to the ideal value. So, the evolved solution works best under tight tracking which is achieved through the control system which drives the offsets towards zero. This behaviour coincides with the intention of the use of the relative fitness evaluation method (Equations 3 to 6) during the CGP evolution. In conclusion, the evolved solution as shown in Equation 20 may possibly be found manually, but, the CGP evolution found this solution without any information about the plate geometry, and only the four sensor voltage values.

5. CONCLUSIONS

A novel way to evolve a fault tolerant *generic* solution was established using special training sets. A virtual environment which achieves suitably complex dynamics to allow rich interactions between Shaky Hand and the environment

was carefully constructed. Analytical reasoning behind the evolved solutions was presented. The solutions were applied to a real Shaky Hand system (Figs.13) to confirm the results in practice. Successful runs without failure were achieved. Operation with one sensor failure could not be distinguished from the ideal fully functional case, proving that this CGP approach can be applied to *practical* sensor fault tolerant applications. CGP is a proven, powerful tool for searching for reliable, practical solutions which would be difficult to find manually. In conclusion, this work has produced a final, fall-back system which will provide *safe*, if degraded, performance of a system when all other fault tolerance mechanisms, based upon multiple redundancy of sensors, cease to be available. Both the method of generating the solutions and the solutions themselves are completely novel. This work opens the door to a different and complementary scheme to achieving sensor fault tolerance.

6. REFERENCES

[1] L. Ashmore. An investigation into cartesian genetic programming within the field of art. Final year project 2000, Department of Computer Science, University of Birmingham, 2000.

[2] H. F. S. H. Bäck, T. A survey of evolution strategies. volume 1802 of *Proceedings of the 4th International Conference on Genetic Algorithms*, pages 2–9. Morgan Kaufmann, 1991.

[3] A. Chamseddine, H. Noura, and M. Ouladsine. Sensor fault-tolerant control for active suspension using sliding mode techniques. In *Workshop on networked control systems and fault-tolerant control*, Ajaccio, Corsica, France, October 2005. COSI.

[4] S. DiPaola. Evolving 'portrait painter programs' using genetic programming (Darwinian evolution) and a portrait of Darwin. website, School of Interactive Arts & Technology, Faculty of Applied Sciences, Simon Fraser University. http://www.dipaola.org/evolve/, August 2007.

[5] C. Edwards and C. Tan. Sensor fault tolerant control using sliding mode observers. *Control Engineering Practice*, 14:897–908, 2006.

[6] S. Harding and J. Miller. Evolution of robot controller using cartesian genetic programming. In *Proceedings of the 6th European Conference on Genetic Programming (EuroGP 2005)*, LNCS 3447, pages 62–72. Springer, 2005.

[7] T. Martinek and L. Sekanina. An evolvable image filter: experimental evaluation of a complete hardware implementation in fpga. In J. Moreno, J. Madrenas, and J. Cosp, editors, *Evolvable Systems: From Biology to Hardware. Proceedings Lecture Notes in Computer Science*, volume 3637 of *6th International Conference, ICES 2005*, pages 76–85, Sitges, Spain, September 2005. Springer-Verlag, Berlin, Germany.

[8] J. Miller. An empirical study of the efficiency of learning boolean functions using cartesian genetic programming approach. volume 2 of *Proc. of GECCO, page 1135-1142*. Morgan Kaufmann, 1999.

[9] J. Miller. Evolving developmental programs for adaptation, morphogenesis, and self-repair. *Seventh European Conference on Artificial Life*, 2801:256–265, September 2003.

[10] J. Miller, D. Job, and V. Vassilev. Principles in the evolutionary design of digital circuits – part i. *Journal of Genetic Programming and Evolvable Machines*, 1(2):259–288, 2000.

[11] J. Miller and P. Thompson. Cartesian genetic programming. volume 1802 of *Proceedings of the 3rd European Conference on Genetic Programming*, pages 121–132. Springer-Verlag, 2000.

[12] J. F. Miller, P. Thomson, and T. C. Fogarty. Designing electronic circuits using evolutionary algorithms. arithmetic circuits: a case study. booktitle= Genetic Algorithms and Evolution Strategies in Engineering and Computer Science, year = 1997, pages = 105-131, publisher = Wiley.

[13] K. Slany and L. Sekanina. Fitness landscape analysis and image filter evolution using functional-level cgp. In M. Ebner, M. O'Neill, A. Ekart, L. Vanneschi, and A. Esparcia, editors, *Proceedings Lecture Notes in Computer Science*, volume 4445 of *Genetic Programming. 10th European Conference, EuroGP 2007.*, pages 311–320, Valencia, Spain, April 2007. Springer-Verlag.

[14] N. Wu. Sensor fault masking of a ship propulsion system. *Control Engineering Practice*, 14(11):1337–45, November 2006.

APPENDIX

A. PICTURES OF SHAKY HAND

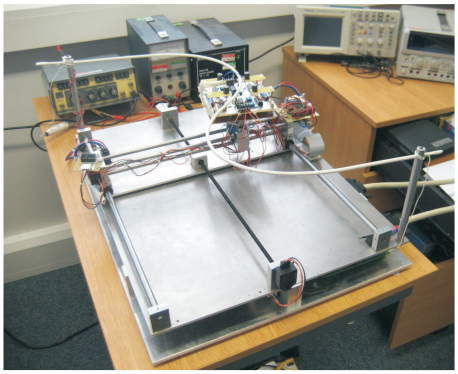

Figure 13: General View of Shaky Hand

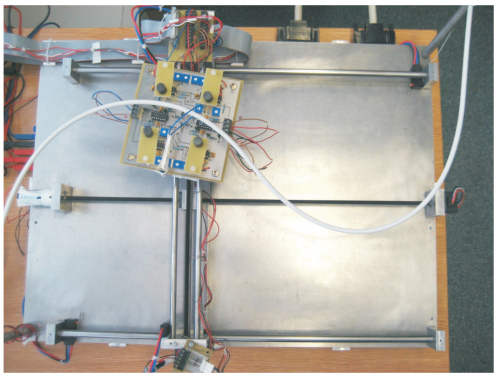

Figure 14: Top View of Shaky Hand