Crane collision modelling using a neural network approach

Ignacio García-Fernández[1], José D. Martín-Guerrero[2], Marta Pla-Castells[1], Emilio Soria-Olivas[2], Rafael J. Martínez-Durá[1], Jordi Muñoz-Marí[2]

[1]Robotics Institute, University of Valencia
E-Mail: ignacio.garcia@uv.es
[2]Digital Signal Processing Group, Electronic Engineering Department, University of Valencia.
E-Mail: jose.d.martin@uv.es

***Abstract*-- The objective of the present work is to find a Collision Detection algorithm to be used in the Virtual Reality crane simulator (UVSim ®), developed by the Robotics Institute of the University of Valencia for the Port of Valencia. The method is applicable to box-shaped objects and is based on the relationship between the colliding object positions and their impact points. The tool chosen to solve the problem is a neural network, the Multilayer Perceptron (MLP), which adapts to the characteristics of the problem, namely, non-linearity, a large amount of data, and no *a priori* knowledge. The results achieved by the neural network are very satisfactory for the case of box-shaped objects. Furthermore, the computational burden is independent from the object positions and how the surfaces are modelled; hence, it is suitable for the real-time requirements of the application and outperforms the computational burden of other classical methods.**
The model proposed is currently being used and validated in the UVSim Gantry Crane simulator.

Keywords: **Neural networks, real-time, simulation, collision detection**

1 INTRODUCTION

The evolution of the hardware and techniques in the field of computer graphics has led to a widespread use of simulators and real-time environments for several purposes. One of the most rapidly growing fields of application is Virtual Reality and driving simulation systems. Such simulation systems are composed of modules that perform different tasks in order to be able to provide the user of the equipment with a feeling of total immersion. The main parts of a simulation system are the visual model, the dynamic model, and the audio-visual and inertia devices, which provide the user with different stimuli.

By means of numerical, mathematical algorithms, the dynamic model calculates the position of the different bodies of the scenario. The visual model has a database that includes all the objects and visual information of the scenario. It uses the state of the simulated system computed by the dynamic model to generate the visual representation of the scene, which is projected by the projection system. To achieve a high degree of realism, an inertia simulation system composed of a mobile platform uses the accelerations computed by the dynamic system to give the user the feeling of movement.

Within this field, the Robotics Institute of the University of Valencia has developed UVSim ® (http://robotica.uv.es/grupos/artec/English/simu.html). UVSim comprises several crane simulators that are currently being used for the training of stevedores working in the Port of Valencia [3]. This is the second most important harbour of Spain in terms of traffic. The simulated machinery, the rubber tyre gantry crane and the quayside crane, are used for the moving and stacking of containers in a harbour terminal, as well as for loading and unloading containers from a container ship. Figure 1 shows some images obtained during a simulation session.

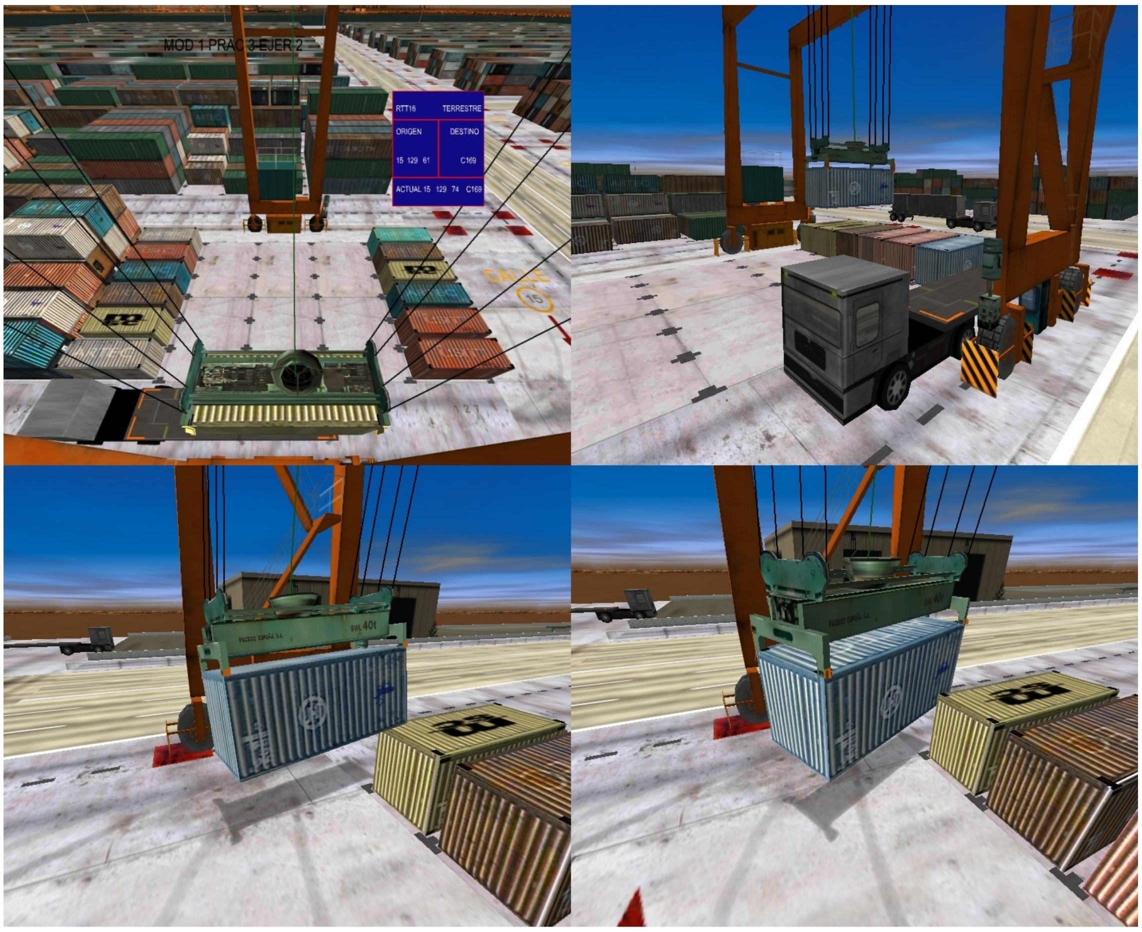

Figure 1. The UVSim ® simulator. Several screens of the simulator software currently being used in the Port of Valencia.

In simulations of this kind, the dynamic models have to calculate not only the dynamic behaviour of the elements of the simulated environment, but also the interactions among them [4,5,9,12]. As it is very important to reproduce the object accurately, the physical interaction between the objects must be taken into account. It is also necessary to have sufficient control of such contacts in order to reproduce the reactions of the system properly. These reactions are created by using forces applied to objects.

The problem of collision handling in real-time virtual environments is commonly dealt with in two major phases. In the collision detection phase, the position and shape of all the objects of the scene have to be analysed in order to determine whether two of them are colliding or not. Then, when a collision between two objects is detected, the contact point and the direction of the forces must be accurately obtained. This second phase is called the collision determination phase [10].

Many algorithms have been implemented [1,6,10,12] in order to obtain both the contact point and the impact direction of the colliding parts. Most of these collision detection methods are based on proximity tests; a proximity test is a set of calculations that gives the distance between two bodies. The distance between two bodies is taken as the distance between their closest points. The precise calculations will depend on the representation of the objects used in each application [12].

A widely used representation of the bodies in a virtual scene is the polygonal representation of their surfaces, because of the flexibility they offer. The surfaces are approximated by a set of polygons that are a linear interpolation of the surfaces. In this representation, the proximity tests are performed between the faces of the polyhedra that represent the two objects. This approach is called the Polygon Soup approach because the representation based on polygons is usually a list of faces without any topological structure. Figure 2 shows an example of the way a complex shape can be split into polygons.

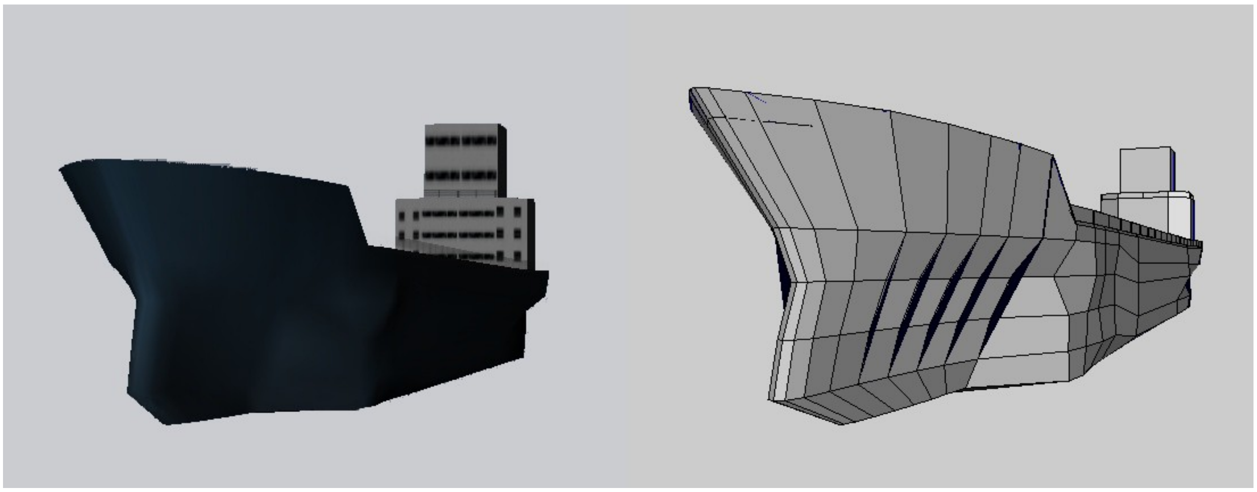
Figure 2. Polygon decomposition. A polygon grid can be used to represent the surface of a complex body in order to use it in computer graphics applications.

Since testing the contacts for all possible pairs of polyhedron would be extremely costly, many hierarchical decompositions of the objects have been proposed [1,7,8,11] in order to accelerate these detections. These algorithms are general purpose methods and, in many cases, obtain a good performance; in the most favourable situations, they achieve a linear or even constant computational cost in relation to the number of polygons.

However, in some applications where a high number of polygons has to be used in order to determine the impact point and direction with sufficient accuracy, many polygons may come into contact at the same time. Then, the computational burden of determining all the contact pairs and calculating an adequate set of contact points will be high, even if hierarchical methods are used. This can make the collision detection problem a bottleneck in a real-time simulation system.

In this paper, we address the problem of collision determination for its application to virtual environments where most of the objects have the shape of a box. We will take advantage of the restrictions imposed by the geometry of the bodies to obtain a model based on Artificial Neural Networks (ANN), which reduces the cost of the collision determination phase.

2 The Problem of Box-Box Collision

Let us consider the problem of contact determination between box-shaped objects. As S. Gottschalk et al proved in [1], the problem of collision detection between boxes can be solved by the calculation of the projection of the boxes along a set of fifteen planes. This involves just a few scalar products of the vectors that represent the axes of the boxes. Despite the simplicity of the contact determination, the problem of obtaining the contact point and the normal direction, is not so simple.

Following the Polygon Soup approach, the surface of each box is split into several triangles [1], for which simple collision detection methods are applied. Figure 3 shows an example of a possible triangle decomposition of a cube, with triangles of different sizes.

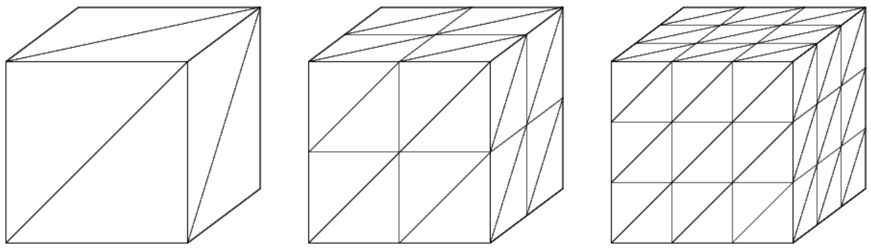
Figure 3. Cube decomposition. The surface of each cube can be subdivided into triangles in order to use Polygon Soup collision detection techniques.

The main problem emerges when the contact point has to be obtained from a list of pairs of triangles that are in contact. If the triangles are too large, say just two triangles per face, the information provided by a set of pairs of triangles is not enough to solve the problem. On the other hand, if we use triangles that are too small, we find two problems: firstly, as the size of the triangle is reduced, the number of triangles increases with the square of the inverse of the size; secondly, in certain situations, the number of triangles colliding at the same time can be very high.

As an example, let us consider a situation in which a vertex of one of the boxes (A) is colliding with a face of the other box (B) with a slight rotation of one object relative to the other (Figure 4). At the time the collision is detected, a small part of the moving object overlaps the other box. The intersection of the surface of both boxes is a triangle on the face of object B, whose size grows as the relative rotation of the objects becomes smaller, involving a higher number of triangles, and, therefore, a higher computational burden.

Also, since there is a wide region involved in the collision, it is more difficult to determine the impact point with the desired accuracy, and further decision techniques must be used to determine the collision point.

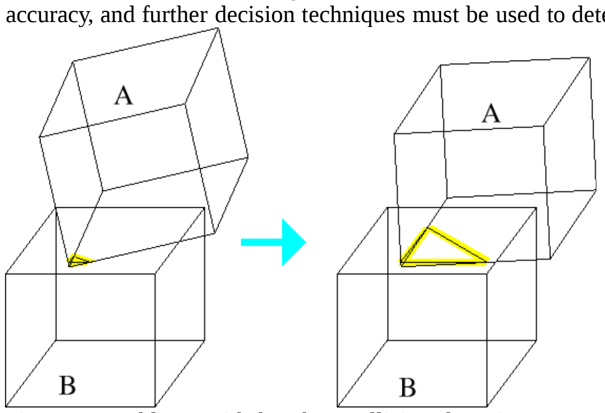

Figure 4. Problems with box-box collision detection. Example of a situation in which the determination of the impact point becomes very difficult when using polygon decomposition of the surface. The shaded region is the region affected by the collision.

3 Artificial Neural Network Approach

The use of Polygon Soups for the box-box impact determination problem has several drawbacks. In order to circumvent some of these drawbacks, we must take into account that many restrictions from a geometrical point of view can be considered; the fact that all the objects have the same shape leads to the idea of finding a model that provides the contact point as an output, with the position and rotation of the objects being the inputs to the model. We propose the use of Artificial Neural Networks (ANN) to solve the problem; we have chosen the most widely used ANN, the Multilayer Perceptron (MLP) for several reasons [2]:

1. This type of models have shown to be appropriate for solving complex, non-linear mappings between two different sets.

2. No a priori knowledge is necessary.

3. This model is capable of finding a relationship between two data sets.

4. The model is built from the data with no assumption about the underlying model to be found. Hence, the neural network is completely defined by the data.

5. In addition, this model is suitable for real-time applications since it has a fixed computational burden once it has been trained.

An MLP was designed to model the relationship between the relative position and the rotation of the two boxes and the contact point. This ANN is composed of individual units known as neurons that constitute the elemental processing unit of the network. (Figure 5).

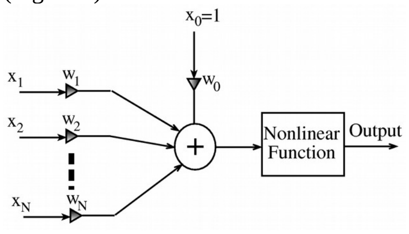

Figure 5. Scheme of an artificial neuron.

Each neuron implements a non-linear transformation of its inputs:

$$y = \phi\left(\sum_{i=1}^{m} w_i \cdot x_i + b\right)$$

where m is the number of inputs to the neuron, w_i are the coefficients known as synaptic weights, *b* is another coefficient known as bias and φ is a non-linear function, typically the hyperbolic tangent. Practical implementation usually considers bias *b* as a synaptic weight whose value is one. The structure is completely defined by taking x_i to be the external inputs, and *y* to be the output of the neuron.

These elements are arranged in layers in which each neuron of a given layer feeds all the neurons of the next layer. The first layer is known as the input layer, and the last one as the output layer. The rest of the layers are known as hidden layers (Figure 6). The number of hidden layers and the number of neurons in each one are parameters which must be adjusted. Due to the complexity of the problem, a network with two hidden layers was developed to achieve the desired accuracy in the modelling [2]. This is because the modelling capabilities of this network come from the hidden layers, where a non-linear mapping of the input space is carried out.

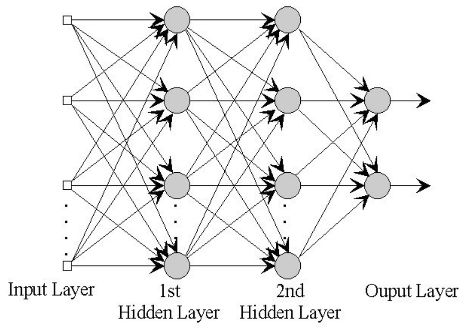

Figure 6. The multiplayer perceptron. General architecture of a multilayer perceptron (MLP) with two hidden layers.

As the problem being dealt with is a continuous output range modelling problem, non-linearity in the output neurons is not necessary (since this non-linearity limits the output range of the neural network). Therefore, the function φ is removed in the output neurons, and only the linear combination of the inputs defines the neuron processing.

Once the structural model is built, it is important to pay attention to the values of the synaptic weights since the behaviour of the system will depend on these values w_i. A supervised learning strategy is carried out in order to find the optimal values for w_i. Supervised learning requires knowing the desired output of the neural network in order to update the weights. Given the error of the network, a cost function is defined so that the minimum of this function corresponds to an error equal to zero [2]. The algorithm used to find this minimum is known as the learning or training algorithm, and the most widely used one is the backpropagation algorithm. This algorithm is based on the delta rule, which consists of considering initial values for the weights and updating them to decrease the value of the cost function (Figure 7).

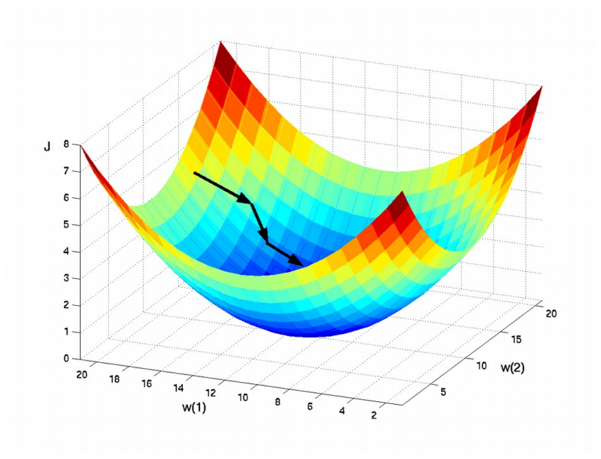

Figure 7. Delta rule (the backpropagation algorithm). A situation with two synaptic weights and a global minimum.

As the gradient of the cost function has the maximum error direction, and the minimum of the function must be found, the learning algorithm should correspond to the following expression:

$$w_k^{n+1} = w_k^n - \alpha \cdot \frac{\partial J}{\partial w_k^n}$$

The most relevant characteristics of the algorithm can be stated as follows:

1. The number of iterations to reach the minimum of the function depends on the initial values of the weights.

2. The minimum achieved may be global or local, depending on the initial values of the weights. This problem is common to all the algorithms derived from the delta rule. In order to circumvent this problem, different values of the initial weights are considered so that different neural models are obtained. Finally, a model is chosen by analysing which weights imply a minimum for the cost function.

3. The value of the parameter α, known as learning coefficient, determines the convergence speed of the algorithm. The larger the value of α, the faster the speed of the algorithm. However, an excessively large value can lead the network to instability.

A modification of this algorithm has been used, namely, the momentum algorithm. It is better in terms of robustness, speed of convergence and stability than the basic algorithm. It consists of adding up the change in the weights of the previous iteration:

$$w_k^{n+1} = w_k^n - \alpha \cdot \frac{\partial J}{\partial w_k^n} + \mu \cdot \Delta w_k^n$$

The parameter μ is known as the momentum coefficient.

Due to the high adaptability of the model, which is capable of determining any relationship between two data sets, the usual procedure is based on dividing the whole data set into two groups, known as the training set and the validation set, respectively. The first set is used to determine the synaptic weights, and the second one is used to test the behaviour of the neural model with data that the network has not yet seen. Since the quantity of data in this problem is very large, small training and validation sets were used. In addition, a large external validation set containing all the remaining data was used.

In order to reduce the complexity of the relationship to be learnt and to reduce the size of the training sets, we benefit from the high degree of symmetry existing in the problem. This symmetry allows us to divide the problem into several smaller ones with analogous solutions: each box can be divided into two trihedra. Then, instead of finding a global model for the collision between two boxes, a more restricted model can be found for the collision between the convex part of two such trihedra (Figure 8).

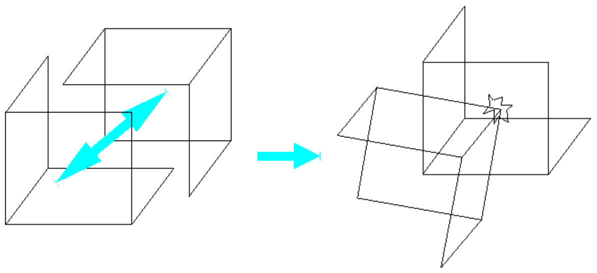

Figure 8. Symmetry. To simplify the problem, each cube has been split into two trihedra and the collisions between them are modelled separately.

The inputs of the model are the relative positions of the two boxes in Cartesian coordinates plus the relative rotation expressed by the Euler angles, commonly used in dynamic models of rigid body systems. The output is the impact point plus a unitary vector that indicates the direction of the collision forces. The length of the vector that gives the impact position can be obtained from its direction, since it must lie on the surface of the cube. Therefore, the output can be expressed by the angles of the spherical coordinates of these vectors. We used a model with six input variables and four output variables. The output ranges were $[-\pi,\pi]$ for the first and the third output unit and $[-\pi/2, \pi/2]$ for the second and the fourth output unit.

The training set was obtained by generating a large set of input positions on the surface of one of the trihedra and considering them as impact points. Then, for each point, each vertex of the other trihedron was placed on that point. The second trihedron was then rotated taking a uniform set of values for the Euler angles with the range $[-\pi/2,\pi/2]$. A similar procedure was followed to obtain patterns for edge-edge collisions. Two boxes of edges with a length of 2 were used to simplify the implementation.

If the mapping that relates the inputs and the outputs of this problem is taken into account, then it is not a continuous mapping. For example, in some cases, a change in the sign of one of the Euler angles can change the impact point from one vertex to another, producing a discontinuity.

To avoid discontinuities in the training set, we divided it into subsets with no discontinuities, and the global function was approximated to each subset. However, we also tried modelling with discontinuities. If the results obtained by the model with discontinuities were not good enough, the use of a committee of several networks would be recommended [13]. Each network would be specialised in a different zone of the input space; these zones should contain no discontinuities.

Over 2,000 networks were trained to tune the parameters of the network (adaptation and momentum constants, number of hidden nodes and range of the initial synaptic weights), thus obtaining the best performance models. The backpropagation algorithm was used to train the network and every training procedure was stopped by cross-validation. In both cases, three sets of data were used: a training set was used to fit the parameters of the model by the backpropagation algorithm, a validation set was used to stop the training by cross-validation, and an external test set was used to obtain a more precise estimation of the error of each model. In the problem with discontinuities, the training set was composed of 8,000 patterns, the validation set of 5,000 and the test set of 75,000. In the problem without discontinuities, the training set was composed of 10,000 patterns, the validation set of 6,000 and the test set of 50,000. The external validation set was considerably larger than the pattern sets used during the training and had no common patterns with the training and validation sets used to provide a more robust modelling.

4 RESULTS

The best results for the model with discontinuities (Table 1) were obtained with an MLP that had two hidden layers; the first layer had twenty units and the second one had eighteen units. The models were evaluated through three performance indexes: the Mean Error (ME) as a measure of bias, and the Mean Absolute Error (MAE) and the Root Mean Square Error (RMSE) as precision measures. Keeping in mind the range of the inputs (the average was over 4.25), the average MAE in a single pattern was almost 10 % of the range, and no bias in the modelling was observed at the ME. It is worth noting that the errors made were very similar in the four outputs of the neural network. This result is good enough to perform a response after the collision. Figure 9 shows the modelling achieved for the first of the four outputs in part of the external validation set. The solid line represents the desired output, and the dotted values represent the corresponding outputs provided by the model.

Table 1. Best results obtained with an MLP for the model with discontinuities.

	Training	Cross-validation	External validation
ME	0.0332	0.0259	-0.0005
MAE	0.4161	0.4383	0.5624
RMSE	0.6947	0.7363	0.8667

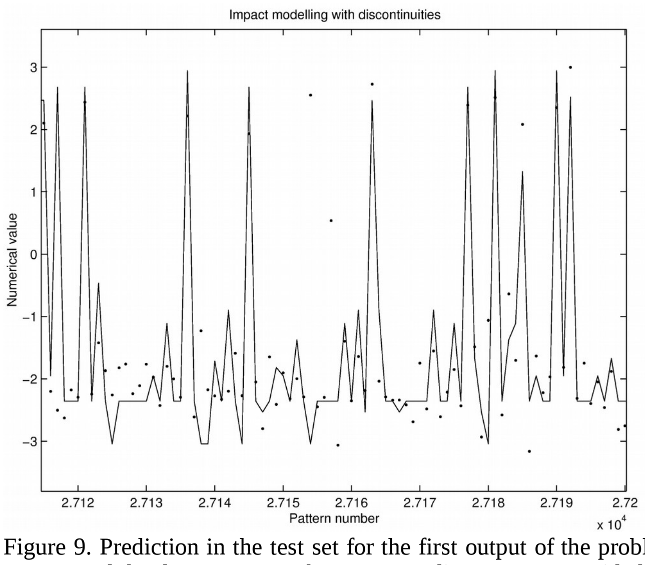

Figure 9. Prediction in the test set for the first output of the problem with discontinuities. The solid line represents the desired output, and the dots represent the corresponding outputs provided by the model.

For the model without discontinuities, the results were obviously better. The architecture of the best network was an MLP that had two hidden layers; the first layer had twenty units and the second one had seventeen units. The results obtained are shown in Table 2. Once again, the errors made in every output were balanced, the average MAE was well below 10 % of the range of the outputs and no bias appeared in the modelling.

Table 2. Best results obtained with an MLP for the model without discontinuities.

	Training	Cross-validation	External validation
ME	-0.0008	0.0031	0.0467
MAE	0.2765	0.2883	0.4523
RMSE	0.5410	0.5748	0.7302

5 Comparison with Polygon Soups

In order to test the performance of the new method, we need to compare the results obtained with the ANN model and those obtained with the commonly used Polygon Soup approach. It would be desirable to have an analytical bound for the cost of the usual method so that we could compare this performance in a precise manner. However, this bound is not available, as all the analytical estimations of the cost are done for simple triangle or polygon tests [1]. The final cost greatly depends on the number of triangles that are in contact with each other.

In spite of this, we did some numerical tests with the purpose of demonstrating how our proposed method surpasses the Polygon Soup approach. The numerical tests consisted of a set of simulations in which both methods were fed with the patterns of the external validation set. The results show that for the box-box collision problem, the ANN is faster than the Polygon Soup approach.

For the Polygon Soup approach, we used the RAPID® software which was provided by the authors of the method. This software computes the contacts between two objects that have been previously split in triangles. The output of the program is a list of pairs of triangles that are in contact. After this software is applied, an additional algorithm needs to be used to obtain the contact point and normal direction. As has been stated above, this problem can be hard to solve in situations where many triangles are in contact with each other. However, once it has been determined that the accuracy of the proposed method is within the requirements of the scope of our application, the only point to be tested is the computational burden of the new method compared to the traditional one. For the Polygon Soup approach, we tested the first step (that provides the list of contacts) as it is the part of the process with the highest computational cost. Indeed, if one method can be proved to be faster than a single part of the Polygon Soup method, then it will clearly be faster than the Polygon Soup approach.

During the simulation, we recorded how long it took for both methods to provide an output. The time estimations were done using the tool *gprof* [14] using a Pentium IV® at a speed of 1.5 GHz. Obviously, the cost of the Polygon Soup approach depends on the number of triangles the surface of each box is divided into. However, as we stated above, after detection, an estimation of the contact point must be obtained from the list of triangles. Thus, the size of the triangles will influence the accuracy that can be reached. If small triangles are used, the accuracy is higher, but so is the cost. The test was repeated for different sizes of the grid, as size is what determines the number of triangles that compose the faces of each cube.

Table 3 shows the results for the case of the RAPID software. The first column indicates the number of divisions made in the edges of each cube. The number of triangles that represent the cube appears in the second column. The value is obtained in the following way: if an edge has N divisions, a face has N^2 squares per face, and $2N^2$ triangles per face. The third column indicates the average time spent to process a pattern by the collision detection routine of the software library in milliseconds, using the test set of 75000 patterns. The time spent by the proposed method is fixed, as it uses a trained network and it takes around 0.5 milliseconds to provide the contact determination.

Table 3. Number of triangles per face and average time spent in the computation of the intersection test for different values of N.

N	Triangles/face	Time (ms)
08	768	1.10
10	1200	1.47
12	1728	1.85
14	2352	1.94
16	3072	2.13
18	3888	2.31
20	4800	2.72
22	5808	2.90

However, the average cost of the process is too high for a real-time environment. In the case of real-time applications, the most common way to test the cost of a process is to study it in the case of maximum load in order to know whether it will perform properly in the worst case scenario. Throughout this paper, we have insisted on the fact that the Polygon Soup approach may have some limitations for real-time computation of contacts between boxes, as there are cases where the list of colliding triangles is large. In order to demonstrate this, we also recorded the number of contact pairs that took place in each contact of the pattern set and the cost of the polygon intersection test. These tests showed that, in many cases the number of triangles colliding was very high, and the cost of the algorithm was multiplied by 10. Table 4 indicates the divisions per edge for the Polygon Soup approach, the maximum number of intersection tests and the cost of the detection process. Figure 10 shows the results of the tests.

Table 4. Time spent by the pattern involving the highest number of tests within the large validation set for each value of *N*.

N	Max. test	Time (ms)
08	11233	11,52
10	14323	15,47
12	17911	19,2
14	21789	23,13
16	24667	24,63
18	26517	28
20	30475	32,44
22	35411	36,77

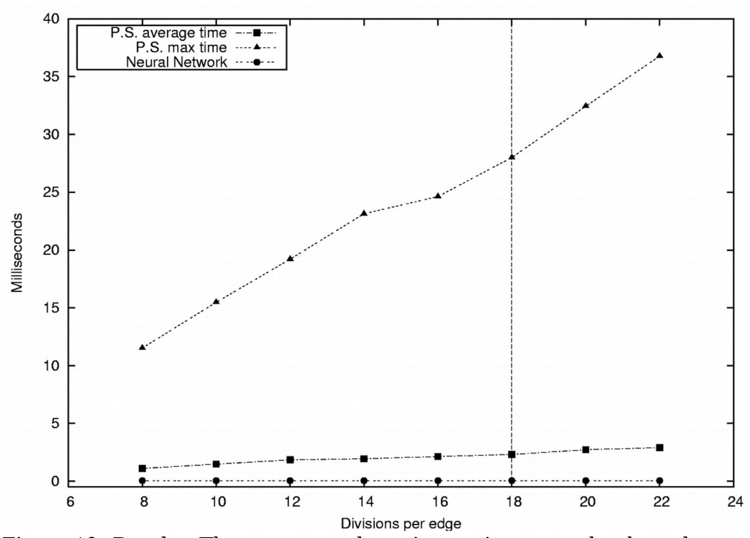

Figure 10. Results. The average and maximum time spent by the polygon soup approach, together with the time spent by the ANN approach, in milliseconds, is plotted for each value of *N*. The vertical dashed line indicates the value of *N* for which the Polygon Soup approach achieves a similar accuracy to that obtained with the ANN.

These cases happen when the two boxes are almost parallel, which is a very common occurrence in crane management. These results show that despite the fact that the mean cost of the Polygon Soup approach is controlled, peaks can appear in the load of the system. This can make it inappropriate for box-box interactions in applications in which the collision detection has to be done many times per second, such as in dynamic systems integration, where the number of iterations per second can be around one thousand. On the other hand, the proposed method based on ANNs has a constant computational cost, which is much more appropriate for real-time applications.

6 CONCLUSIONS

This paper has presented a new approach to the problem of collision detection which is suitable for real-time simulations of environments where most of the objects have a similar shape. On one hand, the results obtained indicate that the impact point and direction have both been obtained accurately. On the other hand, the method has a fixed computational cost. This is because it does not depend on the number of polygons used to model the objects, but rather only on the number of units of the neural network used in the model. In addition, the well-known versatility of methods based on Neural Networks facilitates their application to other environments.

7 FURTHER WORK

Future work is directed towards developing a complete collision detection system that has two phases; initially, a multilayer perceptron or another classifier performs a classification based on both the properties of the relative position of the objects and on their colliding parts (vertex, edges and faces). Then, for each class, the model proposed in this work determines the problem outputs (the impact point and direction). An expert committee of neural networks can provide important knowledge about each part of the input space; thus, the different outputs of the models can be combined, and a more accurate system can be achieved. Once this method is completely implemented, a set of tests can be carried out to compare the results with those of other methods existing in the literature.

8 AKNOWLEDGEMENTS

We would like to thank to the Spanish Society of Stevedores of Valencia (SEVASA) for partial support for this work within the project SIMPORT (Code 3215).

9 REFERENCES

[1] Gottschalk, S., Lin, M. C. & Manocha, D. (1996) OBBTree: A hierarchical structure for rapid interference detection. Computer Graphics, 30 (Annual Conference Series), 171-180.
[2] Haykin, S. (1999). Neural Networks – A Comprehensive Foundation. Upper Siddle River, NJ: Prentice Hall.
[3] Lozano, M. et al. (1999). A reconfigurable projection system for several gantry crane simulators. Proceedings of the Third International Immersive Projection Technology Workshop. (pp. 167-177). Stuttgart.
[4] Milenkovic, V. J. & Schmidl, H. (2001). Optimization-based animation. Proceedings of the 28th annual conference on Computer graphics and interactive techniques, (pp 37-46) Los Angeles (CA).
[5] Redon, S., A. Kheddar, A. & Coquillart. S. (2001). Contact: Arbitraty in-between motions for collision detection. Proceedings of IEEE Workshop on Robot-Human Interaction, Bordeaux.
[6] Thomas, F. et al. (2000). Computing signed distances between free-form objects. Proceedings of the IEEE Int. Conf. On Robotics and Automation, (pp. 3713-3718). San Francisco (CA).
[7] van den Bergen, G. (1999). A fast and robust GJK implementation for collision detection of convex objects. Journal of Graphics Tools, 4(2), 7-25.
[8] van den Bergen, G. (1997). Efficient collision detection of complex deformable models using AABB trees. Journal of Graphics Tools, 2(4), 1-14.
[9] Baraff, D. & Witkin, A. (1992). Dynamic simulation of non-penetrating flexible bodies. Computer Graphics, 26(2), 303-308.
[10] Lin, M. & Gottschalk, S. (1998). Collision detection between geometric models: A survey. Proceedings of IMA Conference on Mathematics of Surfaces, (pp. 37-56). Birmingham. 1998.
[11] Larsen, W. E., Lin, M. & Manocha, D. (1999). IMMPACT: Partitioning and Handling Massive Models for Interactive Collision Detection. Proceedings of Eurographics'99. Milan.
[12] Jiménez, P., Thomas F. & Torras, C. (2001). 3D Collsion Detection : a survey. Computers and graphics, 25(2), 269-285.
[13] Hashem, S. (1997). Optimal Linear Combinations of Neural Networks. *Neural Networks*, 10(4), 599-614.
[14] Graham, S. L., Kessler, P. B. & Mckusick, M. K. (1982). Gprof: A call graph execution profiler. Proceedings of SIGPLAN Notes, (pp. 120-126).