A Smart Surface Simulation Environment

David R. W. Barr, Declan Walsh and Piotr Dudek

School of Electrical & Electronic Engineering
The University of Manchester, Sackville Street Building, Manchester, M13 9PL, United Kingdom
e-mail: david.barr@manchester.ac.uk; declan.walsh@postgrad.manchester.ac.uk; p.dudek@manchester.ac.uk

Abstract – **A versatile simulation system is presented, for observing the effects of different algorithmic and mechanical approaches to physical object manipulation via "smart" surfaces - physical machines which use a distributed array of actuators algorithmically controlled to manipulate objects placed upon them. The system presented is a graphical 3D physics sandbox simulator that allows the modeller to include real-world and hardware related constraints such as inter-object collisions and trajectories, mechanical properties, friction, and sensor/actuator reliability, while conducting experiments in an interactive, repeatable and controlled environment.**

Index Terms – **Smart Surface, Actuator Array, Distributed Manipulator, Bio-inspired robotics, Cyber-Physical Systems, Cellular Automata, Simulation, Physical Model**

I. INTRODUCTION

Smart surfaces, or distributed manipulator arrays, are physical devices that can 'intelligently' manipulate objects placed upon them. Whilst there is no formal definition of a smart surface, they can be characterised by having:

1) *Sensors* - A variety of sensors may be used to determine features about the objects placed upon the surface. These can be global sensors (e.g. overhead camera) or distributed sensors (e.g. arrays of photo-detectors/pressure sensors).

2) *Control* - The features are classified by a control system, typically to identify the object, or to measure various properties of the object to determine its position and orientation. The control system determines a resultant action based upon the state of the surface.

3) *Manipulation* – Coordinated physical manipulation of the objects occurs via spatially distributed actuation (e.g. array of actuators using linear motors, vibration, magnetic levitation, air-currents, etc.).

Smart surfaces are controlled either globally, where all actuators and sensors work under the guidance of a centralised supervisor, or in a distributed manner, where all sensors and actuators are locally self-controlling, sharing information with their neighbours. Typical practical applications of smart surfaces might include the sorting of components, or guided motion of objects via a path determined dynamically in response to other objects trajectories sensed by the surface.

Examples of physical actuator arrays in hardware use various actuation mechanisms including pins, wheels, electromagnetic actuation and air-jet valves. 'Relief' [1] is a tabletop display which uses pin actuation to animate three-dimensional graphics rather than displaying two-dimensional shapes. In a wheel based system described in [2], each cell consists of two rotating wheels with perpendicular axes enabling actuation with two degrees of freedom. Electromagnetic actuation is performed on an interactive workbench of 8 × 8 actuators [3] used to move objects on a tabletop, with the electromagnetic actuators capable of both polarities, allowing the actuators to attract or repel objects of particular polarizations. Further electromagnetic actuation is demonstrated by 'Madgets' [4] in which magnetic widgets such as sliders, knobs and buttons are manipulated on a tabletop to allow user interaction with the tabletop. As well as user input through these widgets, user feedback can be achieved through resistance, vibration or audible feedback, e.g., bell. 'FingerFlux' [5] demonstrates human interaction with a smart surface using electromagnetic actuation by attaching a magnet to a user's hand and causing vibrations or resistance for the user to detect. Distributed control using air-jet actuators

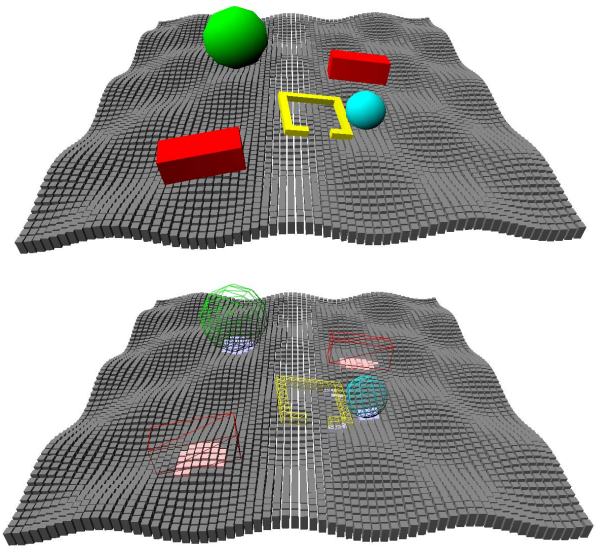

Figure 1 - Smart Surface simulation: (top) default 3D view, (bottom) a wireframe object view with sensor data superimposed onto actuators.

978-1-4799-0652-9/13 $31.00 © 2013 IEEE
DOI 10.1109/SMC.2013.758

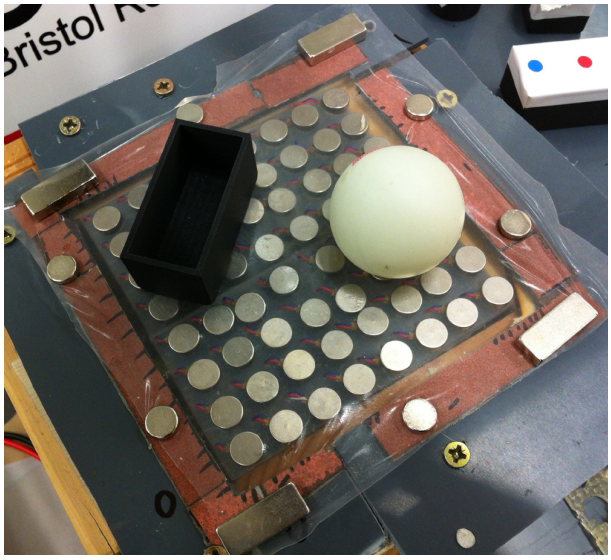

Figure 2 - A hardware distributed manipulator array [7]. Photograph courtesy of Bristol Robotics Laboratory

[6] provides a contactless distributed manipulation in which objects are sensed and compared to objects in a database before being moved to a pre-defined region of the tabletop. Figure 2 shows actuators comprised of oscillators and light sensors, connected via a flexible material where assembled into a small surface, where the creation of waves manipulate the objects [7].

There is a growing research interest in using functionally simple, highly-connected, distributed processing devices to provide computation. Such systems include massively parallel cellular processor arrays [8], cellular automata (CA) [9] and cellular neural networks (CNN) [10]. The study of cellular processing systems involves understanding how locally sourced information can be processed efficiently, in a distributed manner, achieving a globally coherent, task specific outcome. These systems have potentially desirable properties, such as lowered processing and power requirements, component modularity, and no physical limits on the scale of the system (a property a centralised control system would fall foul of). A smart surface is a practical apparatus for the study of emergent behaviour in distributed cellular computing systems. The intelligent control of smart surface actuators based solely on locally available sensory information provides an intuitive experimental platform for the study of the processing systems which underpin the 'smart' nature of the surface.

Smart surface control systems operate using a closed feedback loop [11, 12] of sense, compute, manipulate. The actuators manipulate an object which in turn alters the state of the sensor arrays, which is used to determine the next state of the actuators, thus the physical properties of the actuators, and especially the objects, are important when designing task specific control algorithms. This paper presents software tools which work together to provide a real-time simulation of distributed control algorithms interacting with a physical simulation of the surface, sensors, actuators and the objects to be manipulated.

The physical simulation (Figure 1) uses 'real-world' properties including effects such as inter-object collisions, spacing, timing and response constraints of the physical actuators, and accuracy and capability of the sensors.

The simulation tools are configurable, providing various sensors and actuators, classes of control algorithm, and underlying processing systems. All aspects of the simulation are presented in a highly visual and interactive manner. This paper first describes the simulation tools: SmartSurf, a graphical 3D 'virtual' smart surface, with realistic physical properties and simulation; and APRON, an optimised simulation environment which aids with rapid prototyping and processing of 2D array based data and algorithms. This is followed by demonstrations of the tools being applied to different smart surface tasks with performance benchmarks.

II. SIMULATION APPROACHES

When considering array-centric simulations, it is common to define data abstractions which are idealised, becoming potentially useless when applied to solving real-world tasks and scenarios. Researchers interested in array computation may wish to represent system state, activity and outputs as, for example, 2D arrays (visualised as 2D arrays of pixels) – it then becomes convenient to model peripheral phenomena with similar abstractions that are constrained to work within the capabilities of the simulation tools, e.g. objects might be described as abstract entities, represented as pixels on the 2D grid. System modelling done in this way may be useful to understand the principles of the control algorithms used in smart surfaces but it could be limiting when developing real world solutions. This requires many additional physical properties to be considered, such as object velocities, friction, masses, collisions and inertial and rotational forces. These properties warrant a detailed, physical, 3D model of the system. Manipulating an object implies the object is on top of the surface, requiring simulation of the additional dimension.

For a smart surface control algorithm to be robust, it should be designed and tuned from the ground up in an environment which behaves in a realistic manner. Additionally, being able to visualise the interaction between algorithm and environment provides invaluable insight which will influence subsequent algorithm design decisions.

There are several instances of smart surfaces actually being implemented in hardware [1 - 7]. The development of these devices is constrained by both cost and build complexity, yielding hardware platforms which are often too small in dimensions to capture the wider spectrum of emergent behaviour, and are technically cumbersome with many components prone to failure. The limited availability of hardware also restricts exploring the suitability of smart surfaces (and the wider field of array processing) when applied to physical problem solving. With a virtual 3D physics simulation environment this is no longer the case, and the research can produce control strategies ready to be in place for when suitable hardware arrives.

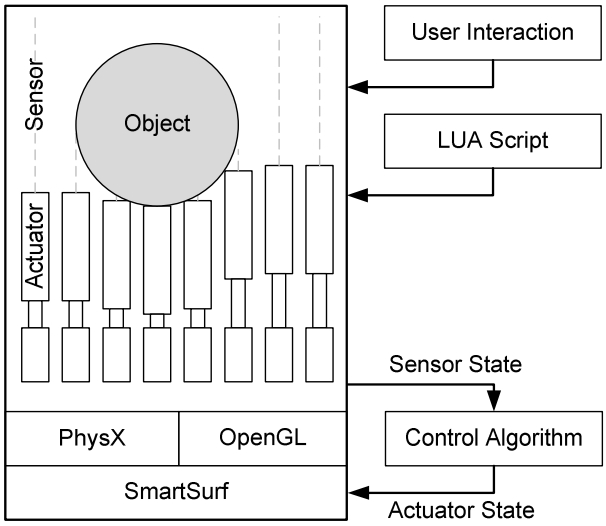

Figure 3 - A schematic of the SmartSurf control system showing "pin" actuators

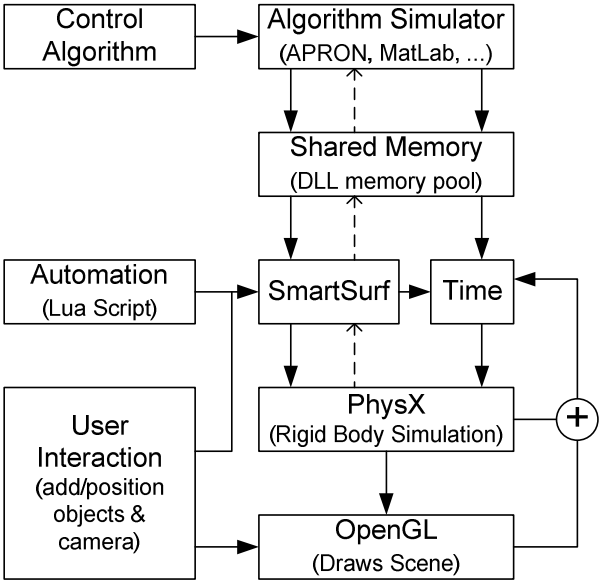

Figure 4 - Data flow and control between different software components

III. SMARTSURF

In this paper we introduce SmartSurf software, a graphical, 3D physics sandbox simulation environment where a smart surface can be described as a matrix of actuators. By default, each actuator is coupled with at least one sensor, this implies that the state of this surface can be described by two distinct vectors, one that describes the state of the actuators, and one that describes the state of the sensors. Even though the surface itself is a detailed 3D model, and there are many properties which define the interactions between the objects and the surface, abstracting to simple sensor/actuator state vectors makes it easy for the control algorithm simulation system to communicate with the surface. For the demonstrations in this paper, it is assumed that the actuators are homogenous and distributed in a grid pattern, though this need not be the case. The control algorithm uses sensory data as input and the actuator state as output (see Figure 3).

Data flow between different software components is shown in Figure 4. Accurate modelling of time is fundamental to realistic physics simulation, and the software presented provides two modes of operation with respect to timing. Firstly, the simulator can operate in 'Passive Mode', where the simulation update is controlled by the performance of the workstation, and its ability to both simulate the physics and render the 3D scene. This mode should be used when human interactivity is a requirement as it presents the simulation in real-time. The control algorithm simulation interacts with the physics simulation when it can. This can lead to odd behaviour when the workstation performance is insufficient for the computational requirements of the simulation, as the perceived speed of the simulation will vary. The second 'Time-Step Mode' only updates time at the request of the control algorithm. The time-step is fixed, ensuring that all updates to the world and from the sensors are coherent. Where this mode gains in accuracy and control, it potentially loses real-time interactivity depending upon the time taken to simulate the physics and the control algorithm. Scripted interactions and data recording will also operate at this time-step. However, on a reasonably powerful workstation this mode will also exhibit interactive behaviour. On a powerful workstation, the simulation may operate in super real-time, which is ideal for scripted interaction but prone to human misinterpretation, as the behaviour of the objects does not 'feel' right and may be counteractive to algorithm development, so an option to limit the simulation performance to real-time (governed by the system wall-time) is provided.

The software is developed using OpenGL [13] and nVidia PhysX [14] software libraries, in C++, that presents the smart surface in a user navigable 3D environment. PhysX is an optimised 3D physics simulation software package that has wide portability, and is often used in the field of rigid, soft-body and particle simulations. Its dominance in the computer gaming industry has yielded a highly optimised, yet flexible library. Although high-precision finite element modelling software does exist, it can be expensive and difficult to interact with in real-time.

Actuators and objects are automatically constructed in a form compatible with PhysX, and are described in terms of geometry and material properties. SmartSurf provides several primitive objects such as spheres, cubes and cylinders for the user to place on the actuator array, but also provides a mechanism for importing custom designed objects (for example from a 3D CAD package), which are converted into acceptable physical models by SmartSurf. Once an object is inserted into the world, its positioning is completely governed by the underlying physics model. Actuators also have physical properties that are used in the simulation, thus the mechanics, material properties and collision detection are handled by the simulator, although their positions are set by the control

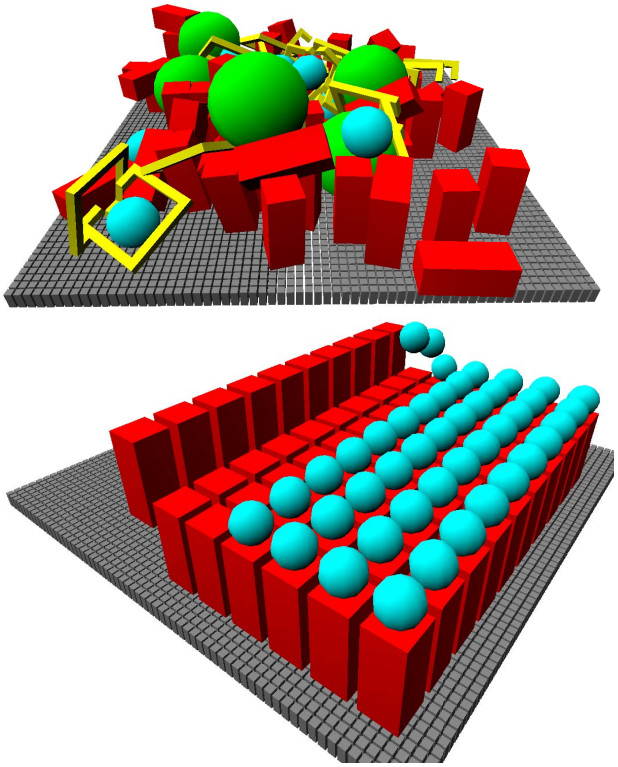

Figure 5 - SmartSurf can handle many thousands of natural (top) and scripted (bottom) object interactions in real-time

algorithm. Sensors are non-physical objects that cannot mechanically interact with anything in the simulated world. The directional proximity sensor for example is represented as the distance from the origin of the sensor, along a line segment until it reaches an intersection point. If no point exists, then there are no objects detected by the sensor. Objects that do intersect can then be queried for other properties such as colour (and even object type) extending the range of possible sensory information available to the algorithm developer.

SmartSurf offers several utilities to aid experimental investigation. Firstly the developer can visually see how well a control algorithm performs, and can tune the sizes of objects and actuators to suit their needs. Secondly, repeatable experiments can be performed by utilising scripts. Scripts (written in Lua [15]) can be written to control how, where and when objects are inserted into the world, demonstrated in Figure 5. SmartSurf can record object positions and orientations over time. The scripting facilities also allow the user to set various environmental properties and some numerical properties of the physics simulation such as integration iterations, and process control. The user is provided with some simple actuators, vertically moving pins, rolling balls, rotating shapes and force fields. Sensory information can be defined as proximity, colour and pressure. The nature of the simulation codebase however makes it trivial to add more complex and custom sensors and actuators.

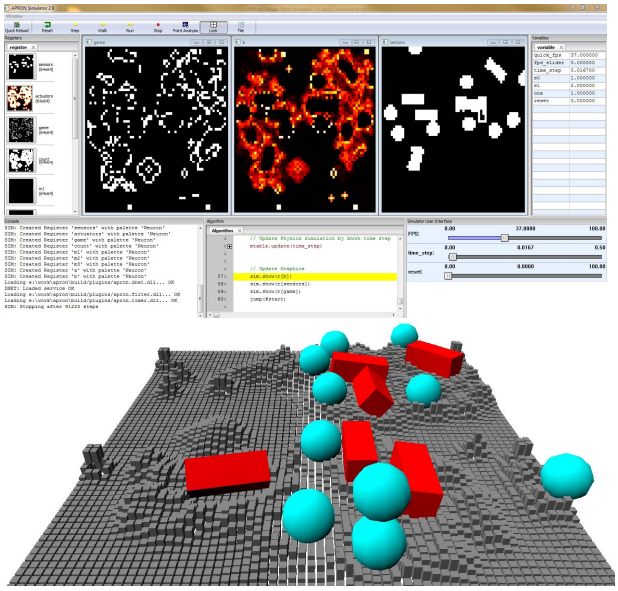

Figure 6 - APRON (top) interacting with SmartSurf (bottom). The state of the surface is visualised alongside other algorithm information

IV. APRON

A bespoke software package called Array Processing Environment (APRON) [16] is used as the control algorithm development and simulation environment. It is optimised for dealing with 2D arrays of data in real-time, and is highly visual and interactive, offering the developer many tools for constructing and debugging array based algorithms. APRON instructions are specifically designed to treat 2D arrays as computational primitives, thus algorithms are described in terms of performing arithmetic operations element wise between arrays. Individual array elements can be 'turned off' based upon a configurable condition including data from the element itself. This offers a local autonomy within the array. APRON provides the ability to easily build extensions to its core instruction set, and libraries of routines which can make interacting with other software and hardware trivial. Although SmartSurf is a stand-alone simulation environment, it can use a shared memory communications layer to rapidly exchange information with other software, and an APRON extension has been built to exploit this. APRON can connect to either the SmartSurf software (see Figure 6), or directly control a smart surface realised in hardware, with the control algorithm needing no conversion between the two. It is predicted that APRON and SmartSurf can be used together to conduct a feasibility analysis into whether or not a particular hardware design effort may be practicable.

V. CASE STUDY

To demonstrate the operation of this software, an array of 64 × 64 actuators is used as the surface for object manipulation. Dimensions are measured in arbitrary units. Each actuator is a rectangular cuboid pin arranged in a two dimensional matrix, aligned to an *x*-*y* plane. The pin is 2 units

long and the top of the pin, which forms the surface, is 0.8 × 0.8 units with a spacing of 0.2 units between neighbouring pins. The pins have one degree of freedom in the *z* axis. Moving the pins either up or down in the *z* axis, manipulates objects. Each actuator is connected to a directional proximity sensor which detects the presence of an object directly above the actuator. By sending a value to the actuator, the simulator displaces the pin to this position. For the purpose of this demonstration, 'down' is the lowest position that is used, i.e. 0 units high, and 'up' is the highest position corresponding to a pin being raised at 4 units high. Due to the laws of physics, an object cannot move from a position of up to down instantaneously in zero time. In order to achieve more realistic and smoother pin motion, the simulator moves pins from their current position to their target position gradually over time, according to a model of the actuator's dynamic capabilities. The border actuators are permanently set to a high value (6 units) to create a wall around the smart surface, preventing the spheres from falling off.

Using this setup, an algorithm is presented which differentiates objects based on their physical properties and separates these objects into two distinct groups. Consider the example shown in Figure 7 where spheres of different sizes lie on top of the surface simulator. The algorithm classifies spheres based on their size and separates them into two groups; the actuators manipulate and position the spheres into two clusters on different regions of the surface. To determine the size of the spheres, sensor data depicts the footprint of the objects on the surface. When the location and size of an object is identified, it can be manipulated by moving the relevant actuators up or down. Local computation determines what action is to be sent to the actuator. Actuators are set to down, if there is an object present, to create a pocket to convey the object. The spheres fall into their pockets and can be manipulated under control. Controlling the actuators results in the pocket being extended either to the west or east of the objects, depending on the sphere's size.

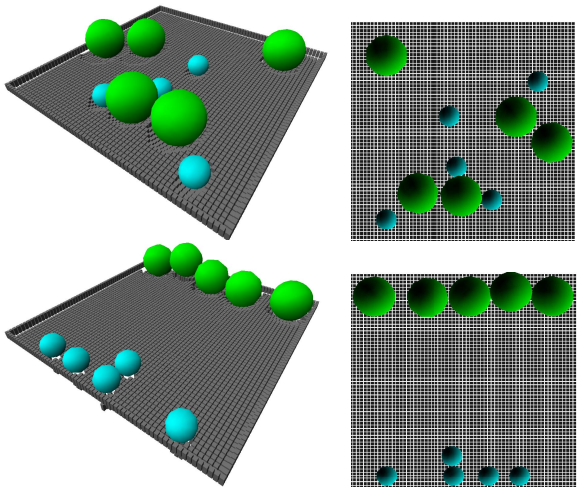

Figure 7 – Simulation of the case study algorithm: (top) initial placement of objects, (bottom) result - spheres sorted by size

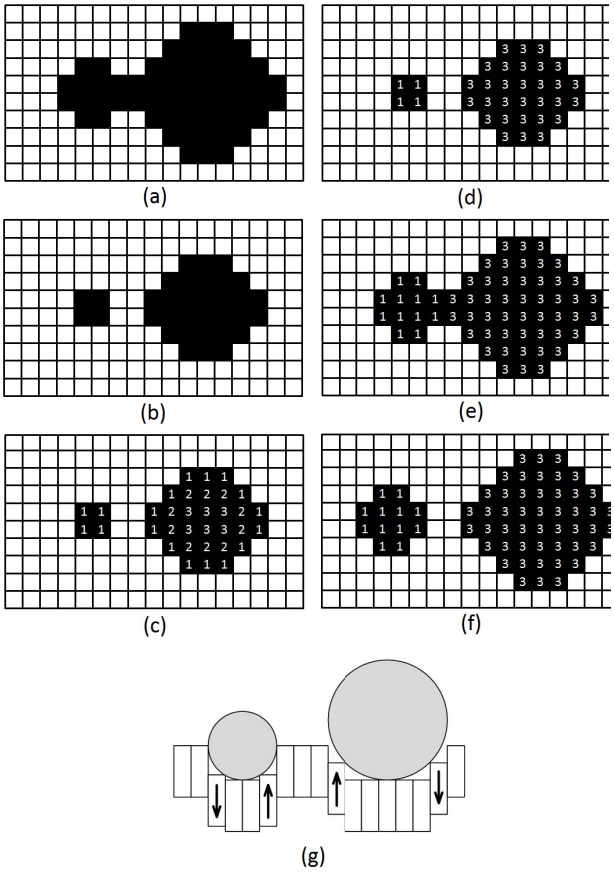

Figure 8 - Different steps involved in the algorithm; (a) sensor data; (b) erosion; (c) count steps to centre of object; (d) maximum number of steps in object; (e) dilation; (f) move west/east; (g) actuator state

The APRON script algorithm directly links a two-dimensional array of 'pixels' to the smart surface, with each pixel having one sensor input and one actuator output. Each pixel in the array is able to communicate with its nearest direct neighbour in the north, east, west and south directions. An object present on top of an actuator and neighbouring activated sensors indicates continuous (or connected) objects (Figure 8a). To avoid touching objects appearing as one continuous object, the control algorithm erodes the sensor 'image' by one pixel (Figure 8b). The sizes of the objects are then determined by counting the number of pixels to the centre of the spheres. This is done by assigning an initial value of '1' to the border pixel data. Then, using the nearest neighbour connectivity, an incrementing value is assigned to all direct neighbours repeatedly until all pixels have been assigned a value (Figure 8c). The largest value indicates the number of steps to the centre of the object. This value is then propagated to all pixels which belong to that particular object to be used as the size identifier for that object (Figure 8d). The object footprint is then dilated by one pixel to cover the complete area of the object (Figure 8e). A threshold value for the size identifier is defined to differentiate which objects move in different

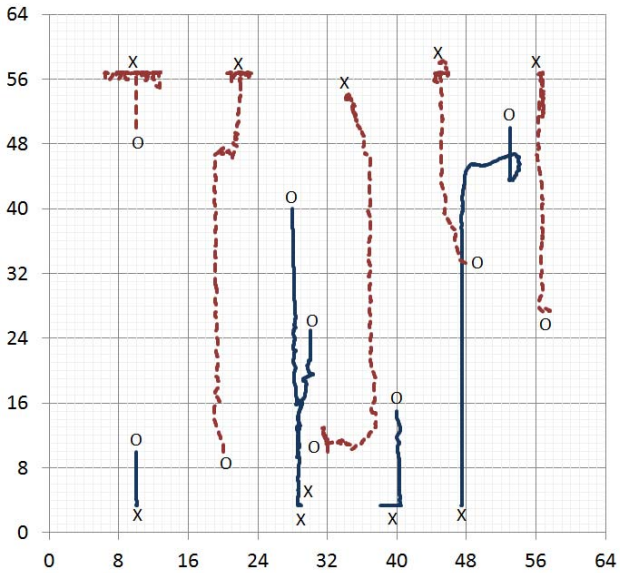

Figure 9 - Trajecctories of spheres during sorting, extracted from Smart Surf, where solid line is the small spheres and dashed line is the large spheres. Initial locations of spheres are marked with 'O', final locations with 'X'

directions. Pixels associated with an object with a size identifier smaller than the threshold (threshold value of 2 is assumed in the example in Figure 8) are moved to the west. Pixels associated with an object with a size identifier larger than the threshold are moved to the east (Figure 8f). The actuators are moved 'up' or 'down' according to the new locations of the object pixels (Figure 8g) which results in objects being pushed laterally across the surface.

As the spheres move to a new resting position (due to gravity) and the sensor data is re-evaluated, the algorithm continues to move the different objects in opposite directions until they reach the raised actuators at edge of the surface preventing them from falling off. It is important to emphasize that the presented algorithm can be implemented in a distributed computing system of locally connected, near-sensor processing units; at no point is a list of objects maintained, instead the information about the objects and their locations is represented by the state of the surface. This algorithm is not affected by object collisions when spheres attempt to move in opposite directions. Due to the erosion that takes place on the sensor data to separate touching objects, the touching objects will continue to push against one another, until one is moved out of the way. The performance on the algorithm can be evaluated by recording the trajectories of all objects in the simulation as shown in Figure 9. The algorithm constantly re-evaluates the sensor data, and eventually the spheres pass each other to form clusters in different regions of the smart surface. This can be demonstrated effectively by placing a large quantity of spheres on the smart surface, as in Figure 7. Even though multiple spheres are placed on the surface and collisions occur, because the sensor data is constantly re-evaluated, and the spheres' physical dynamics are modelled accurately, the spheres can be separated based on their size. The objects being spherical naturally aids in manipulation and collision resolution, a more sophisticated manipulation strategy would be necessary for non-spherical objects.

VI. CONCLUSION

This paper has presented a realistic simulation environment for the development of smart surfaces, alongside software for the real-time interaction, development, visualisation and debugging of the control algorithms used to drive them. Such a suite of tools can be used for requirement analysis when realising a smart surface in hardware, by testing combinations of actuator and sensor designs with task specific goals. An object sorting algorithm has been demonstrated, bringing together the physical and control algorithm simulations, using data from the simulation as empirical proof of its effectiveness. It is hoped that these tools will be adopted by researchers interested in distributed computation, manipulator arrays and other array-centric phenomena.

ACKNOWLEDGEMENTS

This work has been funded by the EPSRC, grant number EP/H023623/1.

REFERENCES

[1] D. Leithinger and H. Ishii, “Relief: A scalable actuated shape display”, in Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction (TEI '10), Cambridge, MA, 2010, pp. 221-222.
[2] J. E. Luntz, W. Messner, and H. Choset. "Distributed manipulation using discrete actuator arrays". The Int. Journal of Robotics Research, 2001.
[3] G. Pangaro, D. Maynes-Aminzade and H. Ishii, “The actuated workbench: Computer-controlled actuation in tabletop tangible interfaces”, in Proceedings of the 15th annual ACM symposium on User interface software and technology (UIST’02), Paris, France, 2002, pp. 181-190.
[4] M. Weiss, F. Schwarz, S. Jakubowski, and J. Borchers. "Madgets: Actuating Widgets on Interactive Tabletops", UIST '10, pages 293-302, New York, NY, USA, 2010.
[5] M. Weiss, C. Wacharamanotham, S. Voelker and J. Borchers, “FingerFlux: Near-surface haptic feedback on tabletops”, in Proceedings of the 24th annual ACM symposium on User interface software and technology (UIST’11), Santa Barbara, CA, 2011, pp. 615-620.
[6] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, and N. Le Fort-Piat, "Distributed control architecture for smart surfaces", IEEE Int. Conf. on Intelligent Robots and Systems, 2010.
[7] I. Georgilas, A. Adamatzky and C. Melhuish, "Towards an Intelligent Distributed Conveyor", Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics), vol. 7429 LNAI, pp. 457-458.
[8] A.Lopich and P.Dudek, "A SIMD Cellular Processor Array Vision Chip With Asynchronous Processing Capabilities", IEEE Transactions on Circuits and Systems - I, vol 58, issue 10, pp. 2420-2431, October 2011.
[9] I. Georgilas, A. Adamatzky, and C. Melhuish, “Manipulating objects with gliders in cellular automata”, IEEE International Conference on Automation Science and Engineering (CASE), pp. 936–941, IEEE, 2012.
[10] L. Chua and L. Yang, "Cellular Neural Networks: Theory," IEEE Trans. on Circuits and Systems, 35(10):1257-1272, 1988.
[11] S. Skachek, A. Adamatzky, and C. Melhuish, “Manipulating planar shapes with a light-sensitive excitable medium: Computational studies of closed-loop systems” Int. J. Bifurcation Chaos, vol. 16, no. 8, 2006.
[12] K. Boutoustous, G. J. Laurent, E. Dedu, L. Matignon, J. Bourgeois, and N. Le Fort-Piat, "Distributed control architecture for smart surfaces", IEEE Int. Conf. on Intelligent Robots and Systems, 2010.
[13] OpenGL, http://www.opengl.org, May 2013.
[14] nVidia PhysX, http:/developer.nvidia.com/physx, May 2013.
[15] Lua, http://www.lua.org, May 2013.
[16] D. R. W. Barr and P. Dudek, “Apron: A cellular processor array simulation and hardware design tool” EURASIP Journal on Advances in Signal Processing, vol. 2009, p. 3, 2009.