Hybrid World Object Tracking For A Virtual Teaching Agent

William Newman, David Franzel, Takeshi Matsumoto, Richard Leibbrandt,
Trent W Lewis, Martin H Luerssen, David M W Powers, *Senior Member, IEEE*

***Abstract*—Fast algorithms and heuristics for real-time object recognition and tracking have enabled a new hybrid world technology in which one can manipulate a real world object and have its virtual world counterpart move correspondingly. This technology has been developed as part of a teaching head platform that was initially designed for language teaching but is now also being used in a range of health-oriented contexts. In this paper, the requirements of the technology are motivated and elucidated, with direct comparison of our proposed heuristics with well known object recognition algorithms**

I. Introduction

VIRTUAL TUTORS are an exciting new development in Artificial Intelligence that have been enabled by a broad spectrum language and learning based technologies.

Whilst speech technology and face generation is based on conventional and commercial technologies dependent on artificial neural networks and related models, the language and computer assisted instruction technologies depend on ad hoc pattern matching, and tend to lack grounding in the sense of connection to the real world.

Children learn language and ontology together, that is they learn about the world and learn to talk about the world in parallel in an intimately connected way. The necessity of grounding in the real world has been extensively argued from the perspectives of psychology, linguistics, philosophy, connectionism and learning theory [1,2], and has been the basis for cognitive linguistic programs in Machine Learning of Natural Language (MLNL) [2,3]. An essential tenet of this approach since the birth of both Cognitive Linguistics and Connectionism is that the nature of the world dictates the nature of the learning mechanism and its biases, including the natural networks of neurons that implement both perception of the world and learning of linguistic concepts, speech, grammar, etc.

Whereas conventional conversational agents use technologies that are intrinisically heuristic, and resemble a text editor more than a learning system, learning approaches with connection to the real world can start to learn grounded language directly [4]. On the other hand the traditional approaches remain more powerful and well developed with a pedigree that extends back to and has not developed much since Eliza [5], and we are using Alice [6] for much of our current Teaching Head research [7] as its AIML language is very easy to teach to both teachers and students who wish to extend the capabilities of an Embodied Conversational Agent.

If grounding is important for language learning, it stands to reason that it is also important for language teaching, and indeed we are borrowing the multimedia technology of MLNL to enhance the capabilities of Computer Aided Language Learning (CALL) [7]. The basic structure of a Thinking Head system is illustrated in Fig. 1: the animated face acts as teacher and illustrates important aspects of correct linguistic, cultural and social usage; to provide a partially enclosed physical arena the two screens are typically angled at around 120°; the second screen provides a window into the teacher's world, while the two screen-based cameras provide different (near orthogonal) viewpoints on the partially enclosed physical arena; the physical arena represents the learner's world and in practice has also been expanded to include half the room, including a whiteboard; and the middle camera concentrates on tracking the learner's face, monitoring their lip movements, expression and facial gestures.

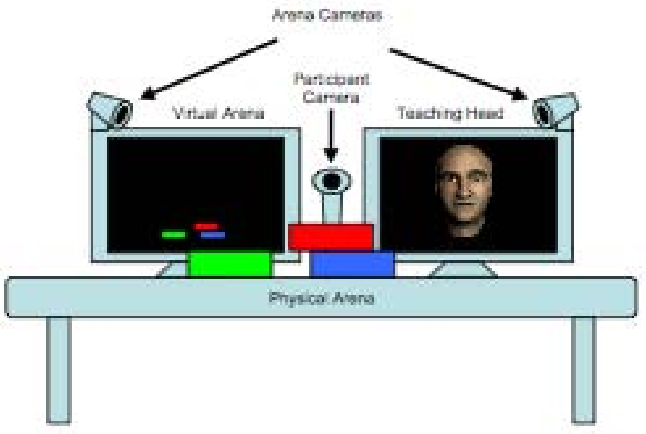

Fig. 1. The layout of the Teaching Head installation. Reproduced by permission of DMW Powers and T Lewis©

Manuscript received February 7, 2010. This work was supported in part by the joint NHMRC and ARC Special Research Initiative on Thinking Systems grant number TS0669874 'Thinking Head'. Please address correspondence to David.Powers@flinders.edu.au.

W. J. Newman and D. Franzel undertook this work as students in the School of Computer Science Engineering and Mathematics, Flinders University of South Australia (email: wjn@internode.on.net, franzel_d@hotmail.com).

T. Matsumoto, R. Leibbrandt, T. W. Lewis and M. H. Luerssen are research fellows in the Artificial Intelligence Laboratory of the School of Computer Science Engineering and Mathematics, Flinders University of South Australia (email: {Takeshi.Matsumoto, Richard.Leibbrandt, Trent.Lewis, Martin.Luerssen}@flinders.edu.au).

D. M.W. Powers is Professor of Computer Science and Director of the Artificial Intelligence Laboratory in the School of Computer Science Engineering and Mathematics, Flinders University of South Australia (email: David.Powers@flinders.edu.au).

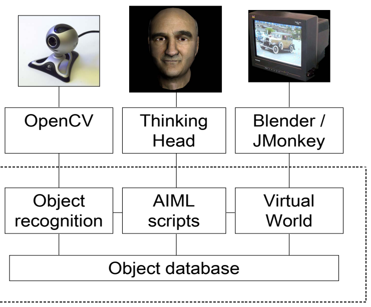

Fig. 2: The layout of the Teaching Head

The Teaching Head concept has since been expanded to include a range of other applications including teaching social skills to children with disabilities, and assisting older people with independent living, but we focus here on CALL.

II. Architecture

The focus of this paper is the specification and development of the visual object recognition and tracking system, and its integration into the world to allow the corresponding objects in the virtual world to be moved and the teacher to discriminate a correct or incorrect answer. The complete description is described at greater length in a thesis [8].

The project consists of four components, as seen in Fig. 2:

- The *object database* contains details, recognition signatures and 3D models of all the objects used in the system.
- The *object recognition* system takes input from the camera, and determines the identities and locations of any objects visible.
- The *virtual world* system takes the details of the objects detected by the object recognition system and displays them in a 3D environment.
- The *Thinking Head* is used as in interface to the user. It gives the user instructions and for language learning applications, will talk to the user in the language that the user is to learn.

The object database is a plaintext file which the other components of the system read upon startup. The object recognition program writes to this file when adding new objects to the system. Other changes, such as renaming or removing objects, can be carried out by hand using a text editor.

The virtual world acts as the central server for the rest of the system. It connects to the object recognition system via networking sockets (through the local loopback address, 127.0.0.1) to obtain the object location data, and sends information to the Thinking Head by means of the Head's API.

The system as a whole, including the hybrid world object tracking system, has been demonstrated using two sample lessons, one involving building towers with blocks, and the other involving manipulating a tiger against a jungle background. The objects were scanned into the virtual world using a Next Engine 3D scanner, but we have also created objects using Blender for printing with our 3D printer.

III. Object Recognition

The visual processing was largely carried out using OpenCV [10] which provides a comprehensive array of basic learning, neural network and statistical tools as well as setting the standard for visual processing tools.

A. *Scale Invariant Feature Transform (SIFT)*

The first version of the object recognition system was built using David Lowe's Scale Invariant Feature Transform (SIFT) algorithm, building on an implementation in C and OpenCV [10-12].

This implementation was successful at locating detailed objects facing the camera, as seen in Fig. 3, but the SIFT algorithm experienced difficulty searching for less detailed objects such as the blocks and toys intended for to use in the project. It also failed to locate the object if it was rotated more than approximately 30 degrees away from the camera, as expected, and patching over this difficulty by using multiple images proved to be a complex task. This was important, since forcing the user to constrain the objects to a particular orientation would be impractical and render the system unusable.

There was also a major issue re speed. On a reasonably modern computer (Intel Core2 Duo, 2.2GHz), the SIFT feature search took 0.5 to 1.0 seconds, before any additional processing. This meant that when the user moved an object, it took around a second for that motion to be reflected on the virtual world display – too slow to be practical.

We determined that the system could be sped up somewhat by using SIFT to identify the objects, then using the much faster Lucas Kanade optical flow tracking method [11] to track the movements of the identified objects while the next SIFT scan was running. However, this would be a complex system to train, and we believed we could do as well with a simpler approach.

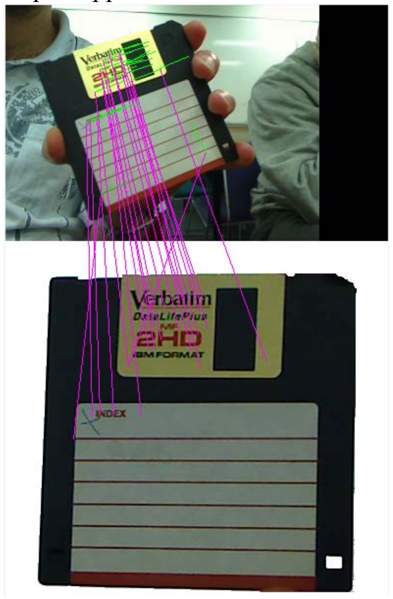
Fig. 3: SIFT feature matching in action. The reference image (below) is located within the camera frame (above) by locating common key points, connected by lines.

B. *Colour Histogram Analysis*

A Colour Histogram Analysis algorithm was developed as a faster and less complicated alternative to object recognition algorithms such as SIFT. It works on the principle that each object will have a distinct set of colours which can be used to identify it. These colours are recorded in histograms and used to identify and locate the object later. The overall structure of the algorithm is given in Fig. 4.

The system works by moving the images into a colour space in which it is easy to separate colour information from lighting information so that the effects of lighting differences can be easily ignored. Information about the colours of each object, summarised into histograms, is retrieved from the database. Next, a *detector* algorithm is used with each object's histograms to determine whether the object is present in the image, irrespective of the location, or any other objects. If an object is present, a *locator* algorithm is used to determine its likely location, and a different, more stringent type of detector is used on the area located to ensure it is not a false positive. The location is then converted from 2D screen coordinates to 3D coordinates for use in the virtual world environment.

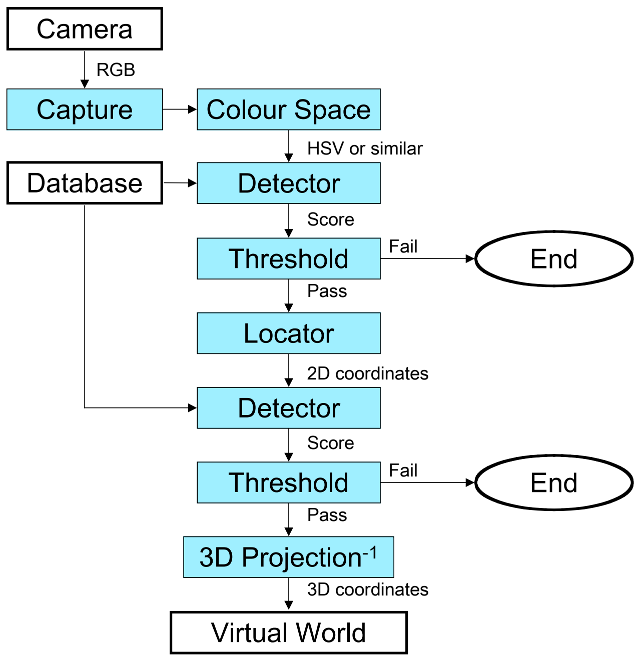

Fig. 4: Colour Histogram Analysis flowchart

1) *Colour Spaces*

The canonical image produced by the camera is in the RGB colour space – that is, there are three bytes for each pixel, describing the intensity of the red, green and blue components of the colour. This is not particularly useful for object colour analysis, since the apparent colour will change due to lighting effects, and this changes the value of all three bytes. In other words, the lighting information and the colour information are encoded into the same variables. In fact colour television and image compression use variants forms that extract some kind of monochrome luminance or brightness signal that is essentially a weighted average of the primary RGB colour signals. This kind of system is potentially also useful for separating the lighting effects from the colour of the object. Several such systems are potentially useful, including variants on YUV and YCbCr that used fixed weightings on RGB, have some relationship to the complementary colour processing of the human visual system, and are used to provide a high information monochrome image. The HSV and HSL represent brightness in terms of the strongest primary component (HSV), or the average of the strongest and weakest component (HSL), rather than an average assigning non-zero weights to all components as in YUV and derivatives.

The Hue (H) in these systems is the same and represents a single recurrent dimension which is very convenient for distinguishing colours in a natural way. Saturation (S) is quite different, with HSV defining a linear Saturation that reduces as one adds white (RGB with equal weight), and HSL defining a linear Saturation that increases as one adds white or black, or alternately as one adds or subtracts white, from a mid-grey base level. Perceptually, for natural scenes, more information goes into the Y signal than into either L or V, whilst more information goes into S in HSV than HSL, and into L versus V. U_{YUV}, V_{YUV}, Cb, Cr, S_{HSV} and V_{HSV} tend to have less information and appear softer or blurry, as can be seen in any comparative review of colour spaces [12].

The HSV colour space was used because it is designed to separate colour information from lighting information, it is very simple and fast, and it appears to maximize the information in HS and minimize the perceptual information in V, which retains most of the shadow. Conversion is straightforward and we used the routines included in OpenCV. Investigation into other colour spaces is still desirable, and proper white balancing systems should also be investigated, noting that the colour of shadow is dominated by sky blue and the colour of direct daylight is dominated by the complementary yellow of the sun. Traditional whitebalancing is towards sunlight, with the effect that shadows tend to come up blue in HSV – this can be construed as a bug or a feature of our system, however it is one that is not so much of a problem indoors!

Colour spaces can be changed without affecting the rest of the system, although object signatures would need to be retrained in the new colour space.

2) *Gathering Training Data*

To train the system to recognise an object, colour data is gathered from many pictures of the object, posed at different locations, rotations, and so forth, and stored in histograms. For the sake of speed and convenience, this is done 'on the fly', meaning that training images are taken from a live camera stream triggered by the user's keypresses, meaning that the user can position the object, take some pictures, reposition the object, take some more pictures, and so on. Using this procedure, many tens to hundreds of samples worth of data can be taken in a matter of minutes, with very little processing time required.

Of course, when camera images of an object are used to gather colour data, the object is not the only thing in the images. There will always be some other colours in the background, and it is important not to include these pixels in the object signature histograms. Therefore, the object needs to be separated from the background. This is accomplished by taking a reference image, without the object in place, and generating a mask by comparing the reference image with each camera frame. The pixels which differ in value should then all belong to the object, which was added to the scene since the reference image was taken.

In reality, there is a certain amount of noise in the camera image, so a threshold is applied to the difference calculation. Practical experimentation shows that a value of 50 (out of 255, for an 8-bit pixel) is a good threshold for removing camera noise, (even in sub-optimal lighting conditions, which tend to increase noise,) whilst retaining most of the object. Since a colour image has three channels, an average of these is used.

3) *Detection: 'Is the object in this bitmap?'*

The first step in the object recognition system is to determine which objects, if any, are present in the camera image, irrespective of their locations. For this, we desire a function which will process the image and return a single number representing how likely it is that the object is present. A threshold can then be used to make a decision. If the object is present, further analysis will be performed to determine its location. This should be a fast routine, as it will be run against every relevant object in the database.

a) Statistical Correlation

One approach to comparing an image to the recorded object signature is to generate a histogram signature from the camera image, then compare that signature mathematically with the object's signature. This can be done by performing a sampled correlation between the two histograms.

b) Fast Unique Pixel Count

Signature correlation (section a)) gives a good indication of whether or not the selected bitmap contains only the object in question, but any other colours in the bitmap will decrease the correlation. In many applications, the object will only make up a small portion of the bitmap, with the rest being other objects and the background. Since the object may only be a small portion of the total bitmap, it will be overwhelmed in the signature by the colours of the background, and return a bad correlation. Therefore, the correlation function does not give a good indication if an object is present.

In order to determine whether the bitmap contains the object, an algorithm was devised which ignores unwanted colours, and the proportions of colours present, and simply checks to see if all the colours in the object are present in the image. It only counts one pixel of each colour, so proportions are ignored, and an object taking up a small window of a large bitmap will still be detected. Colours that are in the image add to the score proportionally to how often they occur in the *training* data.

As well as ignoring proportions and backgrounds, this algorithm has an additional advantage over the correlation function: speed. Correlation requires three multiplications and three additions per pixel, whereas the pixel count requires one bit test, and possibly one addition per pixel. Considering how many pixels must be analysed when processing even a low resolution video stream, this is a significant performance gain.

Further speedups could theoretically be gained by stopping the loop when all the flags are set, since once that occurs the score cannot change any further. However, in practice, it is very rare for an image to contain pixels from every single colour histogram bin, so the speed gained would be negligible.

This algorithm is used to scan the entire image several times, once with each object signature in the database, to determine which objects are present. When the score is high enough to indicate that an object is probably present, more sophisticated (and slower) methods are used to determine the location of the object, and confirm that it is actually present.

Despite the effects of noise, practical experimentation shows the Fast Unique Pixel Count proves to be a very reliable method for determining the presence or otherwise of an object in the camera image. It also suffers only small effects from changes in lighting, changes in the position and angle of an object, and so forth.

The algorithm can return false positives quite easily, especially if many similar colours are present in other objects, but later tests in the locating step check for false positives and filter them out.

The acceptance threshold is different for each object. A reliable method for pre- calculating the threshold has yet to be determined. InIn the stead of making such a calculation, thresholds are currently set manually through observation and experimentation.

4) *Location: 'Where in the bitmap is the object?'*

Once an object has been identified within the bitmap, the next task is to determine where it is. That is, we want to find a bounding box which fully encloses the object but is not any larger than it needs to be to do so.

When considering how to accomplish this, the approach which seems most obvious is to take one of the identifying algorithms developed in section 3) and modify it or extend it in some way to analyse a smaller window or sub-section of the camera image, and find windows which score higher than others, 'scanning' the image for the highest scoring areas, which represent the most likely location of the object. Three algorithms were devised, implemented and developed to accomplish this. They are described in the following subsections.

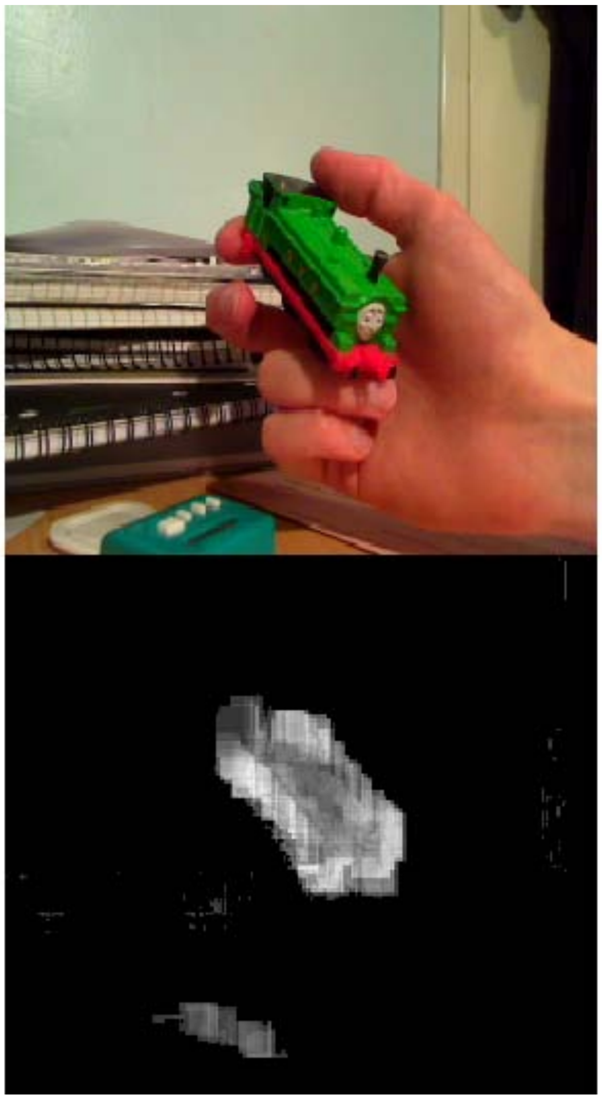
Fig. 5: The sliding window technique, using a Fast Unique Pixel Count filter. The upper image shows the camera image, and the lower image shows the output of the algorithm, searching for the signature of the toy train. The area of the object is clearly marked. Some noise is visible from other objects and details

a) Sliding window filter

One approach to locating the object is to apply the two scoring algorithms to smaller windows of the bitmap, using a sliding window similar to those used with sliding mean and median filters in image processing applications.

The output is a bitmap describing the match quality, where the intensity of each pixel in the output image is proportional to the score of the window around the respective pixel in the camera frame. Areas of high intensity represent the presence of the object.

Since the best response occurs when the window fully encompasses the object, differing window sizes would need to be processed in an arrangement similar to a scale-space pyramid [17,18].

For the correlation function, the single scale-space pixel with the maximum intensity value represents the window which best encloses the object. It would not, however, be necessary to calculate every single point in the pyramid, as the smaller scale scans would be used to rule out unlikely locations and focus the (slower) increasing window size scans only on likely areas.

Practical experimentation showed that this approach was too computationally expensive to be practical, making it too slow for real time video processing. For example, using a window of 16x16 pixels, which is about the minimum practical size, each pixel in the image will be processed 256 times.

b) Image segment and link

To reduce the number of times each pixel is analysed, a variation on window scanning can be used whereby the image is separated into fixed size segments, and each segment is analysed separately. For each segment, a signature histogram is generated, then compared to the object signature using the correlation function, giving the segment a score.

The highest scoring segment is then considered. This segment is combined with each adjacent segment in turn, accomplished by summing their histograms and normalising, and the combined histograms are scored by correlation. If the combined score is an improvement, the adjacent segment is added to a list of segments which describe the object, and itsits adjacencies are tested in the same manner. This process continues until no more segments can be added without decreasing the overall score, at which point all of the segments which contain the object have been identified.

This system operates in a similar manner to the *flood fill* or *boundary fill* algorithms found in many graphics programs, and specifically paint programs.

The problems with this method are twofold. First, since the object will most likely cover several segments, no individual segments contain the whole object, meaning they will not exactly match the object signature. If the object has distinct regions of colour, then most of the segments will describe only some of these colours, skewing the signature, and reducing the difference in scores between a segment of matching object and a segment of anything else – essentially lowering the Signal to Noise Ratio (SNR).

The other is that the segments represent large blocks of the image and do not handle the edges of the object well. AThe score of a segment which partially contains the object will tend to hover around the threshold of inclusion and exclusion from the object, meaning that it is included and excluded intermittently in consecutive frames. When this occurs, it has a dramatic effect on the perceived position of the object, causing a massive amount of noise in the location outputs, even after significant amounts of rolling average or median filtering.

For these reasons, the segment mapping method is not sufficient for the purposes of controlling the virtual world, as all of this noise will be reflected visibly to the user in the virtual world display.

c) Linear Border Search and Refine

The fastest and simplest method for locating the object within a bitmap is to locate each border separately. For most of the process, the Fast Unique Pixel Count is used.

For example, to find the left edge of the object, the image is analysed one column of pixels at a time, in turn from the left edge to the right. Each column is analysed using the Fast Unique Pixel Count algorithm, producing a score for each column. The algorithm is continuous between the columns – that is, the score and flag table are not reset for each column, meaning that the score recorded for each column is actually the score for a rectangle extending between that column and the left hand border, with the full height of the screen.

Over the first few columns the score will quickly climb to a baseline level caused by the pixel colours of the background which happen to coincide with the object signature. There will be a small amount of increase in the score over the width of the image due to different colours occurring. In most cases, this base line will be less than 0.5 (in a scale from 0.0 to 1.0);

When the first columns containing the object are encountered, large numbers of high scoring pixel colours are encountered, causing a rapid increase in score, usually to 0.9 or more. Once these colours have been encountered, there will be little increase in score for the rest of the image.

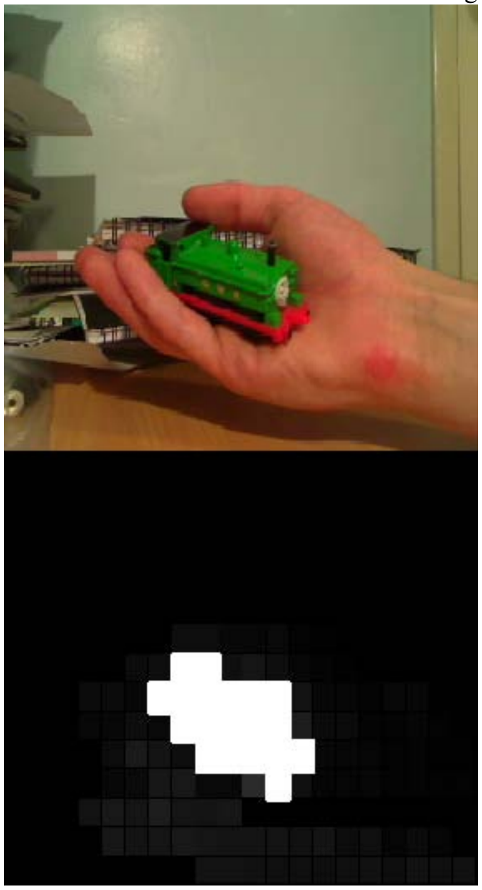
Fig. 6: Image Segment and Link Mapping. Camera image is above, segment map below. On the segment map, each segment is coloured according to its correlation score – brighter shades indicating better matches. The segments whose combined signatures match the object are highlighted by the continuous, lighter shading.

Therefore, the edge of the object can be identified by searching for the largest instantaneous increase in score. This will be the maximum value of the differential of score over columns, calculated by taking a column's score and subtracting the score of the column directly adjacent to it on the left. The greatest difference is taken as the border of the object.

The same procedure is followed for the other three borders. Of course, once the left hand border has been located, the scan for the right hand border need only run from the right edge of the screen to the previously determined left border of the object, since the right hand border will always be to the right of the left hand border.

Once the left and right hand borders have been identified, the scans for the top and bottom borders are performed by scanning the rows of pixels between the previously established left and right borders. These border limits improve the speed of the algorithm by excluding unnecessary pixels from scans for which they are not relevant.

Other objects and background details can cause spurious increases in column scores, but these will usually be significantly smaller than those caused by the object edges. However. small, sharp increases can occasionally overrun the main peak. Some of these can be filtered out by applying a sliding average to the differential signal. The exact benefits and drawbacks of this filtering step are still to be investigated.

Since there is always a large spike of new pixels in the first few rows or columns of a scan, the algorithm allows a small border of 5 pixels (plus the width of the filter mentioned above) in which peaks will not be considered and object borders will not be placed. This stops the initial 'inrush' of pixels from creating the largest differential.

The Fast Unique Pixel Count algorithm has a tendency to generate false positives, so the portion of the bitmap contained within the object boundaries is scored using the correlation function to determine if it really does contain the object, and to filter out any false positives.

This algorithm runs very quickly, and provides accurate and consistent object locations. Often, large errors occur in one border for periods of a single frame, meaning that they can be filtered out by applying a rolling median filter. It is the fastest of the locating algorithms considered, and is also more accurate than image segmentation. The sliding window filter system could possibly produce even more accurate results, but its prohibitive speed limits its usefulness, leaving the boundary search as the best option.

Further development of the border search replaced the single running score with a sliding window, similar to that used in section a). Instead of running the window over the entire image, two 1D score arrays are produced: one vertical, and one horizontal. Each element in the horizontal score array corresponds to the pixel count score for the corresponding column of pixels in the camera image, and the columns surrounding it. The window size used was 16 columns for a 320x240 pixel image. The same process is used with rows of pixels to produce the vertical array.

The two arrays are processed with a threshold, to eliminate noise, and the vertical and horizontal borders were placed to encompass 80% of the total signal 'energy' over the threshold, thus filtering out the occasional score peaks due to noise.

This provided accuracy similar to the sliding window approach at a fraction of the running time, and addressed the tendency for the 'running score' approach to place borders on noise-related score increases.

5) *Inverse 3D projection*

Once the object borders have been located, the 2D coordinates of each object in the image are known. This means that we have the pixel coordinates of a bounding box around the object. In order to use this in any 3D application, we need to translate these coordinates into 3D coordinates *x, y* and *z*. This means applying the inverse of the projection of the 3D objects onto the 2D image [13].

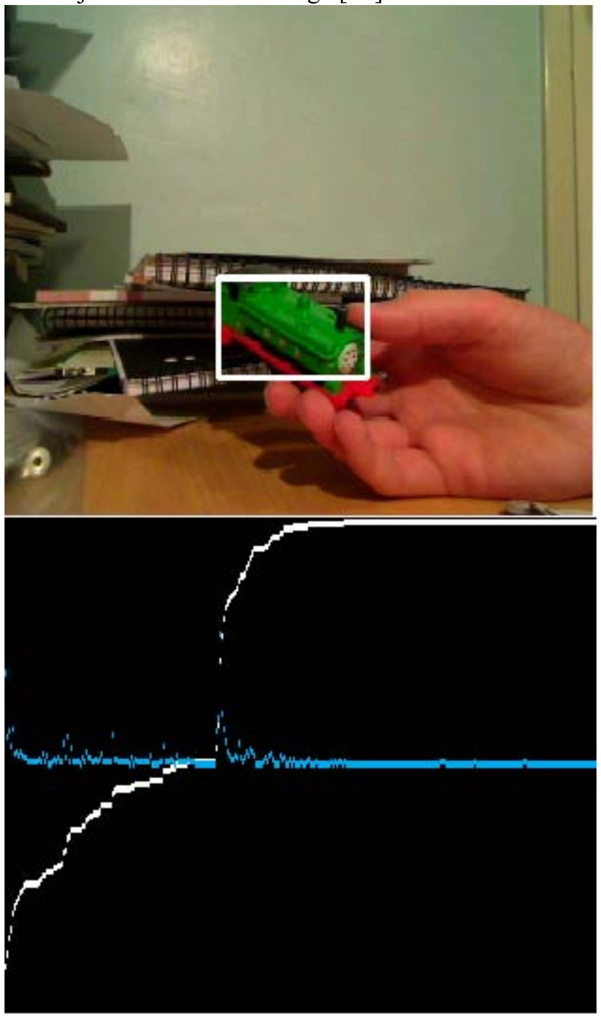
Fig. 7: A practical example. The upper image shows the camera frame with the object borders highlighted. The lower images show the running score (white) and unfiltered differential (cyan-grey) used to locate the left hand border. There is a clear jump which occurs at the left hand border of the object.

IV. Virtual World

The Virtual World displays the objects recognised by the Object Recognition system and reflects their movements. In additional, the Virtual World contains a frame work for creating lessons that are used to teach and test the student's knowledge in the second language of their choice.

This section will discuss the requirements of the virtual world system, how it was developed, and the hardware and software involved.

A. *Virtual World Requirements*

Before we develop a 3D environment for the Virtual World it is useful to consider the requirements the system is required to fulfil.

The main goal of the Virtual World is to reflect objects and their movements from the real world. The first requirement of this goal is that we have models representing the real world objects. These objects needs to resemble the real world objects as closely as possible. Models can be produced by using various 3D modelling software packages. The objects' movementmovements also hashave to also be displayed, but only in terms of translation in the 3D space. Since rotation is not detected by the object recognition algorithm, it cannot be displayed in the virtual world.

As the intent of the Virtual World is to teach a second language to a user, lessons have to be created. These lessons must involve the use of physical objects being manipulated in the real world to achieve a goal in the virtual world. The lessons must also be easy to understand with an interface that adhere to the principles of good interface design, with clear unambiguous relationships between the real and virtual world manipulations. Since the project is being narrated by the Thinking Head, the instructions or any narration given during the use of the system must be clear and logical.

B. *Design of Virtual World*

Before we construct the Virtual World, we produced a design of how we want the system to look and act. To do this, simple drawings were created for every state the Virtual World might enter.

The initial and default state is the 'non-lesson' state, where the objects can be manipulated in any way, against a black background. The heads up display (HUD) displays the names of the objects that are being recognised. The Thinking Head will name objects as they are introduced to the system, and possibly describe their movements.

From the initial state, the Virtual World can enter the 'lesson' state. The lessons are activated through the use of keyboard input. Each lesson includes a background, which is displayed when the lesson state is entered, along with a lesson header to let the user know which lesson he/she is attempting. Narration is provided by Thinking Head, giving instructions for the lesson.

One example of a lesson asks the student to "put the green block on top of the red block" in a traditional blockworld scenario, whilst in a storybook scenario about a tiger, the student is asked to "take the tiger to the lake for a drink" (Fig. 8). Once the user completes the task as given to them by the Thinking Head, a congratulatory message will be given to the user through the Thinking Head, and the Virtual World display.

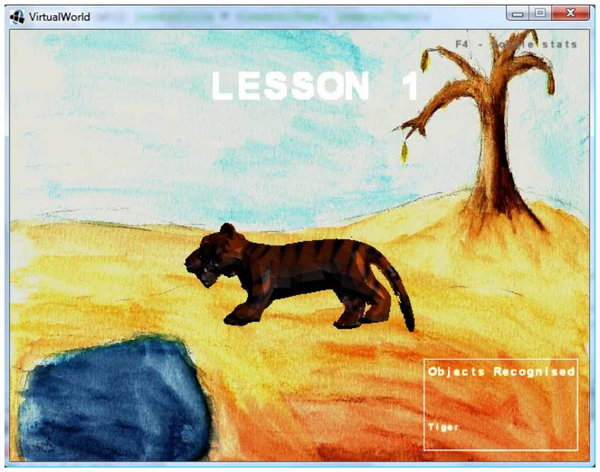

Fig. 8: Screen shot of Lesson 1.

C. 3D API and Modelling

After detailed comparison of alternatives considered as a 3D world software environment and API, we did initially use our own MicroJaea scripting language for Java3D. Unfortunately the development of Java3D has stalled, but due to its speed and its ongoing development backed by an active community, we adopted jMonkey as ideal for this project [13], using it to animate our lessons as illustrated in Fig 8.

Similarly we sought an open-source 3D modeling package to allow us to construct our own objects and adopted Blender [14].

V. Evaluation

In this study, we have considered a variety of algorithms designed for effective object recognition. Our application however, required real-time object-recognition and tracking so our criterion was not accuracy of recognition of objects from some standard dataset, but rather ensuring that it could be used for practical Teaching Head lessons, keeping up with a Logitech Sphere webcam operating at 25fps.

As explained in the detailed discussion of the development of an acceptable package of algorithms and heuristics, we evaluated and discarded the standard Image Recognition Algorithms and sought to develop hybrid algorithms that were more appropriate for our specific application. A detailed summary of the issues associated with the various algorithms is given in Table I, and in particular not that only our Colour Histogram Analysis is fast enough to keep up with a 25fps camera. Moreover, the best competitor, Viola-Jones/Haar, which is better known for a specific face recognition application, is only half the speed at 320x240 pixels (as shown – the gap widens for larger resolutions). In addition, as noted in the table, Viola-Jones has a considerable training overhead which is quite reasonable for a ubiquitous task like face recognition, but quite inappropriate for fast development of lessons by teachers rather than experts in learning and neural nets.

In terms of accuracy, we used whatever office, home, class or desktop the computer was being used in along with props (blocks and other toys) that were selected arbitrarily without any effort to make them distinctive. However, by their nature colourful toys designed for young children do tend to contrast rather well with the university and home environments we tested in. Because we are getting 25 frames per second from our camera, 100% accuracy isn't strictly necessary, but we mainly encountered errors only when most of the object was obscured by a person's hand or other object. By allowing for this and keeping track of the previous history of the object, sufficient stability was ensured for the purposes of this application, notwithstanding the occasional frame in which the image couldn't be recognized. Success was measured in terms of the student being able to complete the lesson without being hindered in any way by the algorithm.

We also demonstrated the ability to distinguish and track multiple objects, and this capability also allows another trick: the Colour Histogram Analysis (CHA) can be applied efficiently to a library that includes multiple views (scales, angles, aspects or parts) of each object, and can thus achieve more reliable detection. The formal evaluation of this potential remains future work, although the basic technique is now being used successfully as part of our entry into the MAGIC Grand Challenge[1], finding and distinguishing hostile and non-hostile objects of interest.

VI. Visual Cognition and Neural Networks

Whereas traditional Artificial Neural Networks train each neuron or synapse separately, all the algorithms we considered train specific Perceptron or RBF feature detectors that are effectively replicated for each pixel at each scale.

Edge detectors, such as the Haar detector, and feature detectors such as the difference of Gaussians (DoGs) used in SIFT are in many ways similar to the visual edge and feature detectors that are well known and fit very general models of visual processing in the ganglia and cortex [17].

The RGB colour space, and variants like HSV, HSL and YUV, all have interesting relationships to human vision processing. The red cone absorption spectrum is precisely complementary to that of haemoglobin, and the green corresponds well to a range of chlorophyll spectra, so that red-green (or red-cyan) opponents (like axes in UV and HS) are good for distinguishing animal from plant as well as features of an animal that is basically filtered haemoglobin. The blue-yellow contrast fits remarkably with a sky-sun contrast and can also identify shadows, whether as unwanted noise/artefact or key textural features [20]. CHA is thus biologically sound as a stage preceding edge detection.

[1] http://www.dsto.defence.gov.au/MAGIC2010/

VII. Conclusion

This paper has compared a number of standard algorithms and well known techniques, and has introduced fast heuristics to enable real time object recognition and tracking. The resulting Colour Histogram Analysis algorithm is fast enough to allow real time Teaching Head lessons. A sample lesson was built and demonstrated at an expo where it was successfully used, in real time, by of the order of 100 visitors.

Future work will need to formally analyse the contribution of the different components of the system. We also may be able to make better use of the possibilities of operating at multiple scales [19,20] and with multiple stored views.

References

[1] D. M. W. Powers, *Special Issue on the Connectionism versus Symbolism Debate*, **THINK 2**:1,1993; **PSYCOLOQUY** 12, 2001.

[2] D. M. W. Powers and Christopher Turk, Machine Learning of Natural Language, Research Monograph, Springer-Verlag (NewYork/Berlin), 1989.

[3] J. Feldman, G. Lakoff, D. Bailey, S. Narayanan, T. Regier, A. Stolcke, 1996. L_0 – The first five years of an automated language acquisition project, **Artificial Intelligence Review 10**: 1-2, April 1996.

[4] D.M.W.Powers "Robot babies: what can they teach us about language acquisition?" Invited chapter, J. Leather and J. Van Dam, eds The Ecology of Language Acquisition, Kluwer, pp.160-182, 2002.

[5] Weizenbaum, Joseph (January 1966), "ELIZA - A Computer Program For the Study of Natural Language Communication Between Man And Machine", **Communications of the ACM 9** (1): 36–45

[6] R.S.Wallace, *The Annotated A.L.I.C.E. AIML.* Accessed 4 February 2010: http://www.alicebot.org/aiml/aaa/.

[7] D.M.W.Powers, R.Leibbrandt, D.Pfitzner, M.Luerssen, T.Lewis, A.Abrahamyan, K.Stevens, "Langauge teaching in a mixed reality games environment," PETRA '08: Proceedings of the 1st international conference on PErvasive Technologies Related to Assistive Environments, pp. 1-7, 2008.

[8] D. Franzel and W. Newman, "Virtual World – Hybrid Reality for Computer Learning and Teaching", Engineering Honours Project, School of Computer Science, Engineering and Mathematics, Flinders University of South Australia, 2008.

[9] OpenCV-2.0.0.0. Accessed on 2nd February 2010 at: http://sourceforge.net/projects/opencvlibrary/,

[10] R. Hess, 2006. *'SIFT Implementation'*, Accessed on 30th October 2009: http://web.engr.oregonstate.edu/~hess/.

[11] D. G. Lowe, 1999. '*Object recognition from local scale-invariant features'*. In Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on, vol. 2, pp. 1150-1157 vol.2.

[12] D. G. Lowe, 2004. '*Distinctive image features from scale-invariant keypoints*'. International Journal of Computer Vision 60:91-110.

[13] B. D. Lucas & T. Kanade, 1981. '*An Iterative Image Registration Technique with an Application to Stereo Vision*'. In IJCAI'81 pp. 674-679.

[14] Illustrations in Wikipedia articles on colour spaces. Accessed on 2nd February 2010 through http://en.wikipedia.org/wiki/List_of_color_spaces_and_their_uses, http://en.wikipedia.org/wiki/YCbCr and http://en.wikipedia.org/wiki/HSL_and_HSV.

[15] Wetherill, J, 2007. '*Comparing Java 3D with jMonkey Engine'*, Sun Microsystems, Accessed on 29th September 2009 at: http: //blogs.sun.com/john/entry/comparing_java_3d_with_jmonkeyengine.

[16] WikiBooks, '*Blender 3D: Noob to Pro/UV Map Basics*', WikiBooks, Accessed viewed 25th September 2009 at: http: //en.wikibooks.org/wiki/Blender_3D:_Noob_to_Pro/UV_Map_Basics

[17] D.M.W.Powers, "Lateral Interaction Behaviour Derived from Neural Packing Considerations", DCS Report No 8317, Department of Computer Science, University of NSW.

[18] T. W. Lewis (2000), "Audio Visual Speech Recognition: Extraction, Recognition and Integration", Flinders University,

[19] T. Lindeberg, 1994. '*Scale-Space Theory in Computer Vision*'. Kluwer Academic Publishers Royal Institute of Technology, Stockholm, Sweden.

[20] T. Lindeberg, 1998. '*Feature Detection with Automatic Scale Selection*'. International Journal of Computer Vision 30(2):79-116.

TABLE I
PROPERTIES OF OBJECT RECOGNITION SYSTEMS USED

	Viola Jones Haarlike	SIFT, SURF	Colour Histogram Analysis
Performance			
Training time and requirements	100-1000+ hand annotated images, many many hours	One picture. A few seconds.	~20 pictures taken 'on line.' 5 minutes or less.
Processing speed	67 milliseconds per frame	680 milliseconds per frame	32ms per frame
Recognition accuracy	Depends on training set	Good for detailed objects	Good for simple coloured objects, occasionally misclassifies.
Location accuracy	Untested	Very accurate	Generally accurate and consistent. Small amount of 'wobble'.
Can the system recognize objects distorted by:			
Translation	Yes	Yes	Yes
Scaling	Yes	Yes	Yes
In plane rotation	No	Yes	No
3D rotation	No	Up to ~30 degrees by affine transform	No
Obscuring	No	Yes. System returns correct location despite obscuring of object.	No. Apparent 'scale' of object is reduced to visible portion, offsetting position and increasing apparent distance.
Limitations			
Mistaken objects	Depends on training set.	Significant feature matching – e.g. writing or symbols.	Predominant matching colours
Limitations	Training time, cannot distinguish rotation.	Slow, some objects do not generate good features, cannot distinguish out-of-plane rotation.	Colours matching between objects, cannot determine rotation, results skewed by obscured portions.