MULTIPLE VIEW CAMERA CALIBRATION FOR LOCALIZATION

Peter B. L. Meijer

NXP Semiconductors
High Tech Campus 31
5656 AE Eindhoven
The Netherlands

Christian Leistner

Inst. for Computer Graphics and Vision
Graz University of Technology
A-8010 Graz
Austria

Anthony Martinière

Ecole Polytechnique
University of Nice
06903 Sophia Antipolis
France

ABSTRACT

The recent development of distributed smart camera networks allows for automated multiple view processing. Quick and easy calibration of uncalibrated multiple camera setups is important for practical uses of such systems by non-experts and in temporary setups. In this paper we discuss options for calibration, illustrated with a basic two-camera setup where each camera is a smart camera mote with a highly parallel SIMD processor and an 8051 microcontroller. In order to accommodate arbitrary (lens) distortion, perspective mapping and transforms for which no analytic inverse is known, we propose the use of neural networks to map projective grid space back to Euclidean space for use in 3D localization and 3D view interpretation.

Index Terms— camera calibration, multiview, sensor networks, neural networks

1. INTRODUCTION

The cost of cameras and wireless networking is coming down to a level where ubiquitous use of cameras as sensors in an ambient intelligence context is economically feasible. In principle this allows for many applications in surveillance and smart environments [1, 2]. Unfortunately, the image processing needed behind the cameras to make good use of the resulting multiple overlapping camera views has until now lagged far behind, both in terms of required computing resources (high GOPS and low power for real-time use) and in terms of proven robust algorithms for extracting meaningful information about the physical 3D reality being monitored - for instance to allow for quickly detecting life-threatening events that require a cascade of follow-up actions.

In recent years, highly parallel SIMD-based (Single Instruction Multiple Data) processor architectures have been shown to allow for low-power high performance pixel processing units that offer a power-efficiency that lies many orders of magnitude beyond that of a typical high-end PC, and at a small form factor [3]. In combination with a camera sensor, these smart camera motes do most image processing close to the image sensor in order to save communication bandwidth and thereby save power in wireless transmissions across local wireless networks. Low-power is an important practical requirement for the ubiquitous application of distributed camera networks. However, much of the energy for short distance transmission of typical live video streams is dissipated in the DA converter of the transmitter, such that high bandwidth video links should be avoided [3, 4]. Key to the application of any such systems in real-life situations remains the calibration or self-calibration of the multiple camera setups.

Complete spatio-temporal self-calibration would be most attractive in maintaining large camera networks, but the problem of fully autonomous calibration under few constraints has not yet been solved. Lee and Aghajan [5] employed Opportunistic Target Observations in their network of smart cameras to perform self-localization, but this often yields relatively high error rates due to the significant view-dependent variations in appearance of the target objects. In situations where self-calibration is not practical or feasible, a simple point-by-point recording of correspondences in mapping Euclidean space locations to associated projective grid locations (locations within the camera views) still allows for an approximate inverse mapping from projective grid locations back to physical locations in Euclidean space. This can be good enough for many applications in home entertainment and smart environments, and it is the approach taken in this paper. Simple low-cost LEDs can be applied as calibration points. A related approach has also been developed by Svoboda [6]. The possibility of using blinking lights such as LEDs to self-localize cameras has been investigated by Taylor and Shirmohammadi [7]. Note that the above inverse mapping as derived from point-to-point correspondences allows for an explicit 3D localization of points that is usually avoided in multiple view interpolation (view rendering) based on the implicit effects of 3D space [8]. Yet an explicit 3D localization is often needed for high-level scene interpretation relating to objects and events in the physical world [9].

The main requirements for good results are that the recorded points span the areas of overlap in camera views, and that the number of points is at least as large as the number of unknown parameters in the inverse mapping model.

1-4244-1354-0/07/$25.00

Many mathematical mapping models can offer adequate interpolation accuracy and numerical stability, but extrapolation tends to be much more risky. Spanning the areas of overlap in the viewing space of the cameras avoids the need for model extrapolation. For numerically stable outcomes it is usually also advisable to have at least a few times more measurements than unknowns to solve for, in a trade-off with measurement effort. Consequently, it is important to limit the number of unknowns by using inverse mapping models with relatively few parameters. We applied artificial neural networks to obtain mapping models, based on the earlier work of Meijer [10]. A related approach using neural networks for camera calibration on standard PCs was proposed by Mendonça *et al*. [11], although at risk of overfitting they seem to have used more unknowns than measurements.

Calibration based on basic point-by-point correspondences forms the basis for a more general dynamic 3D location tracking of multiple feature points for multiple objects in a view, which further requires robust treatment of partial occlusion situations, disambiguation of matching feature points and other view analysis issues that lie beyond the scope of the current paper. This paper will only illustrate the methods by means of a basic 2D to 2D mapping with two cameras, but the very same methods and tools are readily applicable in 3D with two or more cameras, mapping horizontal and vertical pixel positions of all cameras to points in 3D Euclidean space. Using more than two cameras will generally also lead to more robust and accurate mapping models, allow for graceful degradation in case of component failure (such as a broken camera), and can be applied to deal with occlusion effects where not all cameras see the calibration points.

2. METHODS

A 1-point calibration object in the form an a blinking infrared LED is used and positioned at a number of measurement locations. The measurement locations in Euclidean space are more or less uniformly spread out over the entire shared camera view. This is to avoid extrapolation while ensuring a more or less uniform accuracy in the inverse mapping that will be based on these measurement points. The measurement points need not be equidistant nor lie on a grid, because their spatial distribution only serves to control the accuracy of the inverse mapping over the region of interest by weakly anchoring the mapping to the measurement points, for instance through a minimized error in a least squares sense. Automatic background subtraction is performed in real-time by the Xetal SIMD processor, along with determination of the locations of the measurement points. The 8051 microcontroller host handles the low-bandwidth communication in the camera network and performs inverse model evaluation and result I/O.

In general one can only mathematically invert projective mappings under many idealizing assumptions, to avoid that the mathematical analyses become intractable. Moreover, in many cases the various non-idealities of the camera network are not even known in advance, especially if camera network components from multiple vendors are applied and swapped in realistic ubiquitous ambient intelligence scenarios. Therefore, we will instead use general approximation techniques that avoid the need for application-specific mathematical analyses while allowing for arbitrary (lens) distortion, perspective mapping effects and even transforms for which no closed-form analytic inverse exists. Artificial neural networks with training through optimization form one general class of approximation techniques that have been found to work in a wide variety of applications, and we decided to apply these here for calibration purposes in obtaining an inverse nonlinear mapping from projective grid space back to Euclidean space. A related neural network based camera calibration approach was also pioneered by Mendonça *et al*. [11].

To represent the inverse mapping, we used standard multilayer perception networks. It has been proven that these feedforward neural networks can arbitrarily closely approximate any nonlinear multivariate function, although in practice one seeks a trade-off with model complexity and numerical robustness for a still modest number of measurements. This class of networks is a special case of the generalized dynamic neural networks developed by Meijer for behavioural modelling in the time and frequency domain, and the current static mapping was obtained using the same software [10]. In the future, our more general class of dynamic neural networks can be applied to spatio-temporal calibration, to automatically account for network latencies in a distributed smart camera network, to compensate for imperfect timing calibration among the cameras in such the network, to compensate for intrinsic camera capture time shifts as with CMOS rolling shutter cameras, and to model and track time dependent behaviour of physical objects and view signatures observed in (i.e., extracted from) the camera views.

For 2D localization in Euclidean space based on an epipolar plane using two cameras, one has to relate the horizontal pixel position in each of the two camera views to the corresponding two Euclidean coordinate values. A neural network with two inputs and two outputs can accommodate this, and using a 3-layer network with a small number of nonlinear neurons in the middle "hidden" (i.e., non-input, non-output) layer allows for modelling nonlinear dependencies that may include nonlinear interaction terms. A 4-layer network with two hidden layers may be considered in case of more complicated "nested" mapping dependencies, but this is typically a second choice, to be applied only if 3-layer networks of modest size do not suffice (at the expense of efficiency and hence practical value, 3-layer networks always suffice if an arbitrarily large number of neurons in the hidden layer is allowed, while neural networks with more than 4 layers are rarely needed).

Our neural network based modelling software can automatically generate simulation models in a variety of programming and simulation languages, and in this application we

used the automatically generated C code mapping models on the smart camera motes for a "live" mapping of projective grid space back to Euclidean space. For this purpose, the C code mapping models are compiled for the 8051 microcontroller, for subsequent evaluation on one or more of the 8051 smart camera cores. The image coordinates obtained in multiple camera views are communicated among the smart cameras, as needed for actual evaluation of the neural network mapping functions on the distributed smart camera network in operation. This has been tested to work correctly on the physical setup, and the artificial neural networks thus form the "brains" of the smart cameras in mapping projective grid space back to Euclidean space.

2.1. Neural Network Equations

The neural networks used in this paper are defined in this section. A detailed motivation for the various specific choices can be found in [10]. Any time-dependent terms used for modelling timing effects through differential equations will be discarded for the static calibration modelling purposes of this paper (but are applicable to the more general case of spatio-temporal calibration). As a result, the neural networks reduce to conventional multilayer perceptron networks. Layers are counted starting with the input layer as layer 0, such that a network with output layer K involves a total of $K+1$ layers. Layer k by definition contains N_k neurons. A vector notation with bold font is used to denote information on all neurons in a particular layer. A neural network has a vector of inputs $\boldsymbol{x}^{(0)}$ and a vector of outputs $\boldsymbol{x}^{(K)}$.

The equation for the output, or excitation, y_{ik} of one particular neuron i in layer $k > 0$ is given by

$$\tau_{2,ik}\ \frac{\mathrm{d}^2 y_{ik}}{\mathrm{d}t^2}\ +\ \tau_{1,ik}\ \frac{\mathrm{d}y_{ik}}{\mathrm{d}t}\ +\ y_{ik}\ =\ \mathcal{F}^{(ik)}(s_{ik}, \delta_{ik}) \quad (1)$$

where $\mathcal{F}^{(ik)}$ is a (generally nonlinear) function having an optional transition parameter δ_{ik}. The timing parameters $\tau_{1,ik}$ and $\tau_{2,ik}$ will be kept at zero for the static modelling purposes in this paper. The weighted sum s_{ik} of results from the preceding layer is further defined as

$$\begin{aligned} s_{ik} &\triangleq \boldsymbol{w}_{ik} \cdot \boldsymbol{y}_{k-1} - \theta_{ik}\ +\ \boldsymbol{v}_{ik} \cdot \frac{\mathrm{d}\boldsymbol{y}_{k-1}}{\mathrm{d}t} \\ &= \sum_{j=1}^{N_{k-1}} w_{ijk} y_{j,k-1} - \theta_{ik} + \sum_{j=1}^{N_{k-1}} v_{ijk} \frac{\mathrm{d}y_{j,k-1}}{\mathrm{d}t} \end{aligned} \quad (2)$$

for $k > 1$, involving weighting parameters w_{ijk} and v_{ijk}, an offset parameter θ_{ik}, and similarly for the neuron layer $k = 1$ connected to the network inputs $x_j^{(0)}$

$$\begin{aligned} s_{ik} &\triangleq \boldsymbol{w}_{ik} \cdot \boldsymbol{x}^{(0)} - \theta_{ik}\ +\ \boldsymbol{v}_{ik} \cdot \frac{\mathrm{d}\boldsymbol{x}^{(0)}}{\mathrm{d}t} \\ &= \sum_{j=1}^{N_0} w_{ij,0} x_j^{(0)} - \theta_{i,0} + \sum_{j=1}^{N_0} v_{ij,0} \frac{\mathrm{d}x_j^{(0)}}{\mathrm{d}t} \end{aligned} \quad (3)$$

which is analogous to having a dummy neuron layer $k = 0$ with enforced neuron j outputs $y_{j,0} \equiv x_j^{(0)}$, or in vector notation $\boldsymbol{y}_0 \equiv \boldsymbol{x}^{(0)}$. All timing parameters v_{ijk} and $v_{ij,0}$ are also kept at zero for the static modelling purposes in this paper.

Finally, to allow for arbitrary network output ranges in case of bounded functions $\mathcal{F}^{(ik)}$, a linear scaling transformation is added to the output stage

$$x_i^{(K)}\ =\ \alpha_i\ y_{iK}\ +\ \beta_i \quad (4)$$

yielding a network output vector $\boldsymbol{x}^{(K)}$.

The above function $\mathcal{F}^{(ik)}$ is for neuron i in layer k applied to the weighted sum s_{ik} of neuron outputs $y_{j,k-1}$ in the preceding layer $k-1$. The optional transition parameter δ_{ik} may be used to set an appropriate scale of change in qualitative transitions in function behaviour, as is common to semiconductor device modelling, but for the current application a δ_{ik}-independent classic logistic function suffices,

$$\mathcal{F}_1(s_{ik})\ \triangleq\ \frac{1}{1 + e^{-s_{ik}}} \quad (5)$$

Again referring to [10], it can be shown that the above neural network equations can arbitrarily closely approximate any multivariate static function. Under very weak conditions, this approximation property applies even when the nonlinear $\mathcal{F}^{(ik)}$ are the same for all neurons, as with the use of the logistic function, while requiring only three network layers for static models.

3. EXPERIMENTS

In this section we present some experiments on our smart camera set-up, with an emphasis on some of the unique features of the SIMD architecture. As already indicated, the LED is detected using a simple background model of the form

$$B_t(x, y) = \alpha F_t(x, y) + (1 - \alpha) B_{t-1}(x, y), \quad (6)$$

where B_{t-1} is the last stored background image and F_t the current frame.

With our Xetal SIMD processor, each image line is computed in just two cycles, with each VGA line, consisting of 640 pixels, handled as 2 times 320 pixels. The software abstracts away from such underlying image wrapping issues, and represents it by a single instruction, as shown in the software listing 1, line 12.

Listing 1. SIMD background model line

```
while(++current_row < MAX_ROWS)
{
  // Fetch yuv image_row from sensor
  yuv = sensor_yuv();
  ...
  if(BackgroundModelStored())
  {
    // Line of background {t-1} stored in memory
    out[0] = dpram_get();

    // Computing line of New background {t}
    out[0] = yuv[0]*alpha + (1-alpha)*out[0];

    WriteToMemDisplay(out,mem,out,0);
  }
  ...
}
```

Each camera has to locate the infrared LED in the projective space. The SIMD processor computes for each frame the row and column where the LED is detected (see listing 2).

Listing 2. SIMD IR-LED detection

```
1  while(++current_row < MAX_ROWS)
2  {
3  ...
4    // Detection of new moving objects
5    temp = abs(out[0]-yuv[0]) > T ? 1: 0;
6    temp = NoiseReduction(temp5x5);
7
8    // Detection LED line and column
9    if (temp > 0) Detected_Row = current_row;
10   for (i=0 ; i< MAX_COLS; i++)
11   {
12     if (FIRST_PIXEL > 0) Detected_Col = i;
13     temp = temp.neigh(-1);
14   }
15 ...
16 }
17 if (Detected_Row > 0) Row = Detected_Row;
18 if (Detected_Col > 0) Col = Detected_Col;
```

Note that by setting the threshold T (listing 2 line 5) very high, *e.g.* at 80, and using the high infrared sensitivity of the optical sensor, it is possible to assume the LED as the only "non-background" object in the scene.

As a further precaution, a noise reduction filter is applied to remove for example spurious reflections present in front of the cameras. This filter is a 5×5 filter, applied to a binary image. It simply counts the number of bright pixels detected in the 5×5 neighbourhood of the current pixel. If this number is not high enough, we heuristically consider this to be noise in the scene or internal to the sensor (for example due to a low luminosity), and we remove the current pixel from the set of changing pixels detected.

In our case we exploit the high parallellism of the SIMD processor in first detecting the line showing the LED, and we further use the Global Controller (internal to the SIMD processor) to compute the pixel column. The Global Controller can only take into account the value of one pixel, which is why we have to left-shift the detected line in order to analyze the right neighbouring pixel (see listing 2, lines 12 and 13). The advantage is that we need to perform the column detection only once per frame, as soon as the image line showing the LED is correctly detected.

As a result, all the LED detection work is done by the SIMD processor. The final neural network calibration step is readily implemented on the 8051 microcontroller, which allows for using standard ANSI C code.

3.1. Multiple Camera Setup

Our improvised table-top multiple camera setup, illustrative for a quick temporary setup, is shown by the photograph in Fig. 1. For the example discussed in this paper we only used two of the three cameras, as illustrated in Fig. 2.

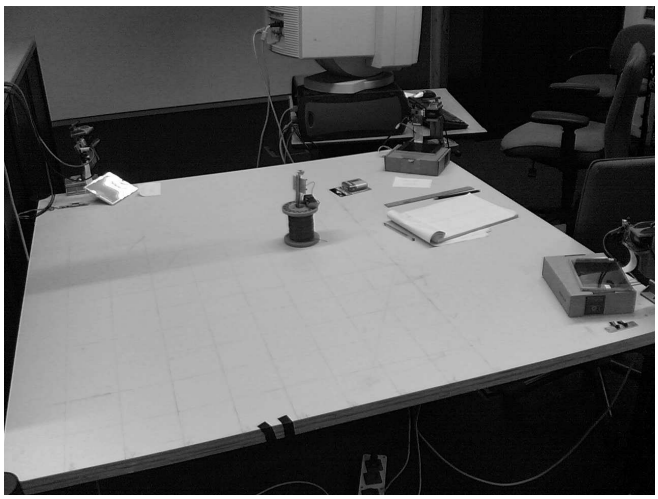

Fig. 1. Photograph of the multiple camera setup, here with three smart cameras at the corners of the table, and with a blinking infrared LED near the center of the table.

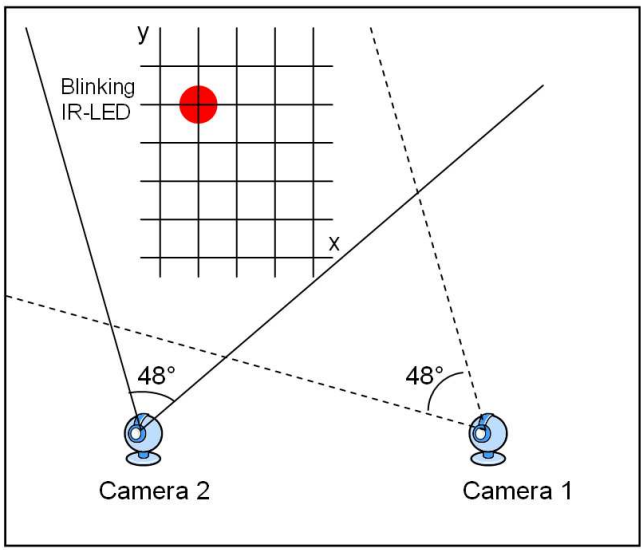

Fig. 2. Schematic of multiple camera setup.

3.2. Results

Through measurements we obtained integer values i_{1m} and i_{2m} for the horizontal locations in the two camera views as a function of the physical locations (x_m, y_m) in Euclidean space, where m represents the measurement counter. This can be described with an unknown vector function $\boldsymbol{F}$

$$\begin{pmatrix} i_{1m} \\ i_{2m} \end{pmatrix} = \boldsymbol{F} \begin{pmatrix} x_m \\ y_m \end{pmatrix}$$

and the aim of the neural network modelling is therefore to approximate the (also unknown, but assumedly existing) inverse vector function $\boldsymbol{F}^{-1}$,

$$\begin{pmatrix} x_m \\ y_m \end{pmatrix} = \boldsymbol{F}^{-1} \begin{pmatrix} i_{1m} \\ i_{2m} \end{pmatrix}$$

via a neural network model. In reality, we will only find an approximate inverse vector function $\hat{F}^{-1}$, and the resulting approximate locations in Euclidean space will be labelled $(\hat{x}_m, \hat{y}_m)$.

Apart from analyzing the modelling error at measurement points that were included in the neural network learning phase, several measurement points were deliberately omitted from the learning phase in order to obtain further indications for the generalization properties of the resulting neural network by afterward evaluating the neural model errors at these excluded measurement points.

Figs. 3 (small dots) and 4 illustrate the grid of Euclidean measurement locations and the measured projective grid locations, respectively. The latter grid clearly appears distorted as compared to the rectangular Euclidean grid, as a consequence of the relatively small and varying distance to the two cameras.

Various neural network topologies were tried, for instance including networks that had two hidden layers between the two outputs and two outputs, using a 2-2-2-2 network topology, but good accuracy was often quickly obtained using neural networks with only 3 neurons in a single hidden layer and 2 neurons in the output layer, that is, using a 2-3-2 network topology for a 5-neuron model as shown in Fig. 5, and using the standard logistic function for the neuron nonlinearity. Best models were selected afterward. Only one of the successful 5-neuron models is further discussed in this paper, for illustration purposes.

The big dots in Fig. 3 show how the selected 5-neuron network maps the measured projective grid points back to Euclidean space. The maximum model error for the reconstructed $(\hat{x}_m, \hat{y}_m)$ over the entire projective grid (i_{1m}, i_{2m}) was 1.35%. This was considered comparable to the manual positioning errors in moving the blinking infrared LED over the Euclidean grid. Note that although a regular grid in Euclidean space was here used for convenience and for illustration purposes, this is not a requirement for the applied methods.

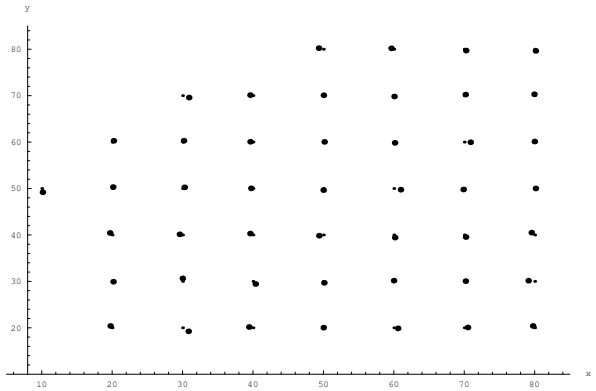

Fig. 3. Euclidean locations $(\hat{x}_m, \hat{y}_m)$, in cm, obtained from the neural network model using the measured projective grid locations for input (big dots). For reference, the original Euclidean measurement locations (x_m, y_m) are included as small dots.

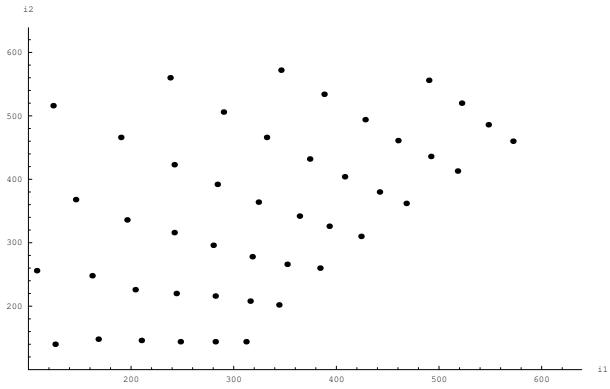

Fig. 4. Measured projective grid locations (i_{1m}, i_{2m}) (integer positions) in the two VGA camera views, obtained with a rectangular Euclidean measurement grid.

3D plots of the neural mapping functions are shown in the Appendix, Figs. 6 (first output $\hat{x}_m$) and 7 (second output $\hat{y}_m$), along with the automatically generated C code in listing 3, where minor formatting changes were applied for use in this paper.

With these satisfactory results, we compiled the automatically generated standard C code model functions for use on the 8051. The setup supports the Zigbee protocol to communicate i_{1m} and i_{2m} for live function evaluation on one of the smart cameras, as needed to display the calculated Euclidean locations $(\hat{x}_m, \hat{y}_m)$ on a remote PC.

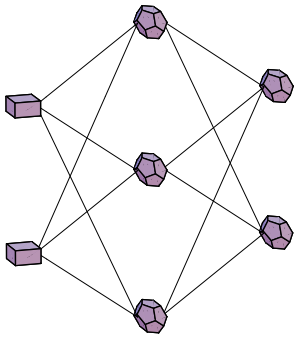

Fig. 5. 2-3-2 neural network with 5 neurons.

4. DISCUSSION

The results show that a calibration based on a more or less arbitrary set of individual measurement points is feasible without detailed knowledge about the camera setup or the camera properties. Application-specific mathematical analyses can be completely avoided. Moreover, the approach can easily account for various non-idealities in the system, such as the often unknown lens distortion. Even with complete knowledge of the system, mathematical analyses would often be too time-consuming for practical purposes, or turn out to be intractable.

In principle, a completely self-contained auto-calibration (self-calibration) without user intervention would appear still more elegant, but this puts additional requirements to the setup, such as the need to have cameras see other cameras. With typical camera viewing angles of only sixty degrees or less (in our case 48 degrees), this is not always practical or attractive, while the least obtrusive camera locations in a room may also occlude their appearance to the other cameras in that room. An elementary manual calibration procedure similar to the one outlined in this paper may in such cases still be necessary.

5. CONCLUSION AND FUTURE WORK

In this paper we showed how a simple 1-point calibration protocol in combination with standard neural network modelling techniques can be applied to obtain a model that maps a projective grid space back to Euclidean space for use in 3D localization and 3D view interpretation. The methods were illustrated by means of a 2D localization mapping with two cameras, but are readily applied to the more general case. Results proved to be highly accurate, and will form a basis for the future development of general dynamic 3D location tracking of multiple feature points for multiple objects in each camera view, with the added possibility, through our generalized dynamic neural network formalism, of performing a general spatio-temporal calibration that accounts for a wide variety of timing effects.

6. ACKNOWLEDGEMENTS

We thank Alexander Danilin for his help in setting up the background subtraction and calibration measurement code.

7. REFERENCES

[1] C. H. Lin, W. Wolf, A. Dixon, X. Koutsoukos, and J. Sztipanovits, "Design and implementation of ubiquitous smart cameras," in *SUTC '06: Proceedings of the IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing -Vol 1 (SUTC'06)*, Washington, DC, USA, 2006, pp. 32–39, IEEE Computer Society.

[2] M. Bramberger, A. Doblander, A. Maier, B. Rinner, and H. Schwabach, "Distributed embedded smart cameras for surveillance applications," *Computer*, vol. 39, no. 2, pp. 68, 2006.

[3] R. Kleihorst, B. Schueler, A. Danilin, and M. Heijligers, "Smart camera mote with high performance vision system," in *Workshop on Distributed Smart Cameras (DSC 2006)*, Boulder, CO, USA, June 2006.

[4] M. Bhardwaj, A. Chandrakasan, and T. Garnett, "Upper bounds on the lifetime of sensor networks," in *ICC - Proceedings of the IEEE International Conference on Communications (ICC)*, 2001, pp. 785 – 790.

[5] H. Lee and H. Aghajan, "Collaborative node localization in surveillance networks using opportunistic target observations," in *ACM Multimedia Workshop On Video Surveillance and Sensor Networks (VSSN)*, 2006.

[6] T. Svoboda, D. Martinec, and T. Pajdla, "A convenient multi-camera self-calibration for virtual environments," *PRESENCE: Teleoperators and Virtual Environments*, vol. 14, no. 4, pp. 407–422, August 2005.

[7] C. J. Taylor and B. Shirmohammadi, "Self localizing smart camera networks and their applications to 3d modeling," in *Workshop on Distributed Smart Cameras (DSC 2006)*, Boulder, CO, USA, June 2006.

[8] Y. Ito and H. Saito, "Free-viewpoint image synthesis from multiple-view images taken with uncalibrated moving cameras," in *Proc. IEEE Int. Conf. Image Processing (ICIP 2005)*, Genova, Italy, 2005, vol. 3, pp. 29–32.

[9] H. Aghajan, J. Augusto, C. Wu, P. McCullagh, and J. Walkden, "Distributed vision-based accident management for assisted living," in *Proceedings Int. Conf. on Smart homes and health Telematics (ICOST)*, Nara, Japan, June 2007.

[10] P.B.L. Meijer, *Neural Network Applications in Device and Circuit Modelling for Circuit Simulation*, Ph.D. thesis, Eindhoven University of Technology, 1996.

[11] M. Mendonça, I.N. da Silva, and J.E.C. Castanho, "Camera calibration using neural networks," *Journal of WSCG - poster abstract at 10-th Int. Conf. Central Europe on Computer Graphics, Visualization and Computer Vision, (WSCG 2002)*, vol. 10, no. 1-3, February 2002.

8. APPENDIX

Listing 3. Generated neural model code (reformatted), as run on the smart cameras (8051 microcontroller)

```
#include <math.h>

double f1(double s) {
   return(1.0 / (1.0 + exp(-s)));
}

void net0(double in[], double out[]) {
   double net0l1n0, net0l1n1, net0l1n2,
          net0l2n0, net0l2n1;

   net0l1n0 =
    f1(-3.8127875846276100e-03 * in[0]
       -4.2295785912067137e-04 * in[1]
       +3.1665623494486885e+00);

   net0l1n1 =
    f1(-9.1588388556664759e-03 * in[0]
       +1.2290676953540203e-04 * in[1]
       +9.0614964706215173e-01);

   net0l1n2 =
    f1(-3.3711715680603675e-03 * in[0]
       +1.2160470931375164e-03 * in[1]
       +1.9435120601542750e+00);

   net0l2n0 =
    f1(+4.3930104113043384e+00 * net0l1n0
       -6.5139698438269900e-01 * net0l1n1
       -7.0793079196186302e+00 * net0l1n2
       -1.2824132753249337e-01);

   net0l2n1 =
    f1(-1.0007183969110965e+01 * net0l1n0
       -2.7076964131170778e-01 * net0l1n1
       +6.7926874876093679e+00 * net0l1n2
       +5.4240606874043866e+00);

   out[0] = -3.3278043362310491e+01
            +7.0023061720757732e+02 * net0l2n0;

   out[1] = -4.8864233148180318e+02
            +6.0016425300491528e+02 * net0l2n1;
}
```

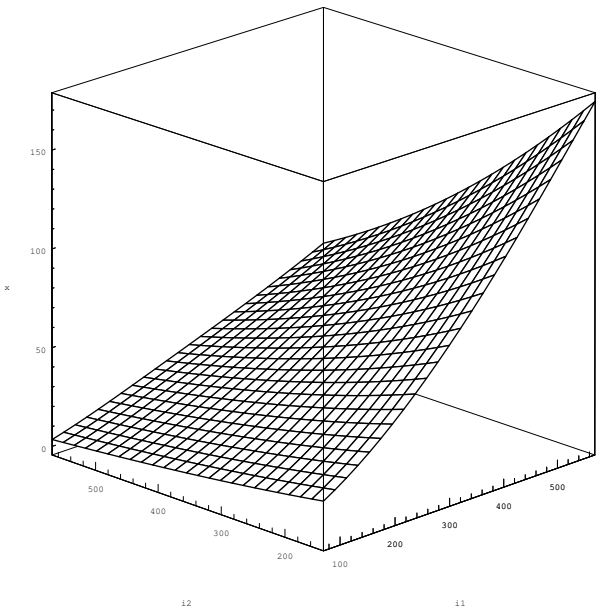

Fig. 6. $\hat{x}_m$, in cm, modelled as a function of (i_{1m}, i_{2m}).

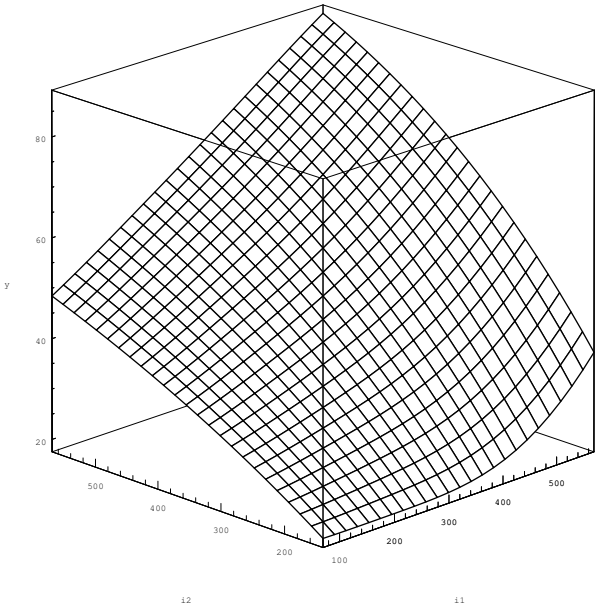

Fig. 7. $\hat{y}_m$, in cm, modelled as a function of (i_{1m}, i_{2m}).