FARLAP: Fast Robust Localisation using Appearance Priors

Geoffrey Pascoe, Will Maddern, Alexander D. Stewart and Paul Newman

***Abstract*— This paper is concerned with large-scale localisation at city scales with monocular cameras. Our primary motivation lies with the development of autonomous road vehicles — an application domain in which low-cost sensing is particularly important. Here we present a method for localising against a textured 3-dimensional prior mesh using a monocular camera. We first present a system for generating and texturing the prior using a LIDAR scanner and camera. We then describe how we can localise against that prior with a single camera, using an information-theoretic measure of image similarity. This process requires dealing with the distortions induced by a wide-angle camera. We present and justify an interesting approach to this issue in which we distort the prior map into the image rather than vice-versa. Finally we explain how the general purpose computation functionality of a modern GPU is particularly apt for our task, allowing us to run the system in real time. We present results showing centimetre-level localisation accuracy through a city over six kilometres.**

I. Introduction

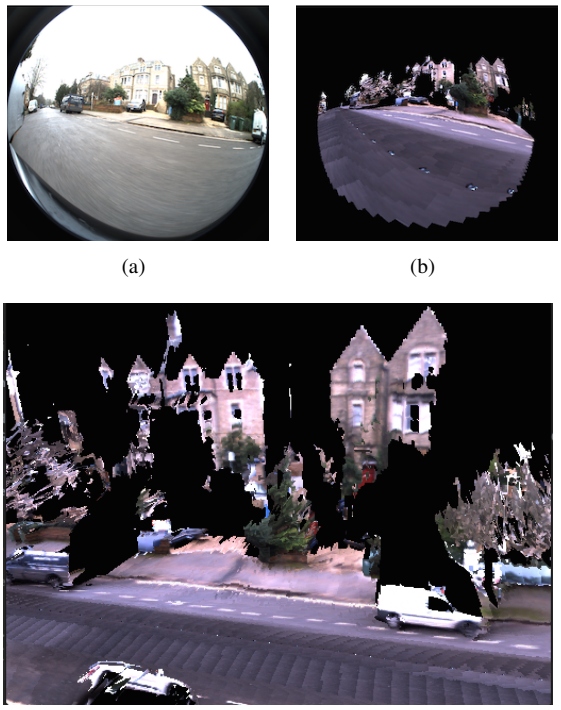
This work leverages commodity GPU hardware to build a large-scale metric localisation system which exploits dense textured scene models of our own making. Using only mono-cameras, the system operates in real-time across marked appearance changes. While such a system can find application in a host of robotics domains, in this paper we focus on its use in on-road autonomy.

State-of-the-art localisation systems [1], [2] for autonomous vehicles rely on high-cost laser-based sensors such as the Velodyne [3]. Whilst these sensors provide high-fidelity information about the vehicle's environment, their cost makes them a challenging proposition for mainstream adoption of autonomous vehicles. We are motivated to explore what can be done with cameras alone at run time, in conjunction with a prior model of the environment.

The approach we advocate is having a small number of survey vehicles equipped with laser sensors, which infrequently visit all roadways, building high-quality maps. By exploiting the information thus made available as a prior, the vast majority of cars on the road would then localise into a map using only monocular cameras. This approach shifts the expensive sensing equipment onto just the few specialist survey vehicles, dramatically reducing the costs for commodity car ownership.

In this paper, we describe a system which uses laser and camera data from the survey vehicle to build fully-textured 3-dimensional meshes of the environment. Along with knowledge of the projective properties of our cameras, this allows us to generate a synthetic image showing what a camera located at any point in the map would see. Equipped with such a camera, we localise by finding the location in the map at which the expected image from the prior most closely matches the live image.

Authors are from the Mobile Robotics Group, Dept. Engineering Science, University of Oxford, UK. {gmp,wm,alexs,pnewman}@robots.ox.ac.uk



(a) (b)

(c)

Fig. 1. Our localisation system relies on generating virtual images from a map consisting of a textured mesh, and comparing those images with the live camera feed. We use a simple look-up-table based distortion model to create virtual images with the same distortion as the live camera lens. (a) Live camera image, (b) projection of the prior using a highly distorted model, (c) a birds-eye view of the prior at the same location.

We make use of normalised information distance (NID) to evaluate the similarity between live camera image and the images generated from our prior. This differs from previous dense localisation approaches such as DTAM [4], in that it does not rely on the actual colours in the camera image matching those in the prior.

We apply non-linear distortion using a look-up table, to distort the mesh before colouring it. This allows us to make use of cameras with very wide fields of view, significantly increasing the size of the region in which the system can converge on the true pose.

The computational power required to generate and evaluate synthetic images is typically very high. Our method, however, is highly data parallel, operating independently on each pixel. We thus make use of the rendering and parallel computation abilities of modern graphics processing units (GPUs) to enable real-time operation.

II. Related Work

The visual localisation system described in [5], [6] relies on the comparison of a sparse coloured pointcloud prior and a live camera image. This system carries out a quasi-Newton optimisation to find the pose that results in the lowest NID between the projection of the pointcloud into the frame of the camera and the live image.

Other systems have been developed that use dense meshes rather than a sparse pointcloud as a prior. It was shown by [4] that a dense approach provides richer information about the scene, and leads to more accurate localisation.

One such system is described by [7]. In this case, the mesh is a commercially available product — this system does not include the creation of the prior. For this implementation, standard mutual information is used as the objective function. Rather than using a quasi-Newton method to approximate the Hessian, the Hessian is calculated directly, although it is only calculated once, and assumed to vary little. Whilst projections of the prior are rendered via OpenGL [8], the rest of the pipeline is carried out on the CPU, and localisations can be performed at a rate of approximately 0.25Hz [7]. Our system, in contrast, exploits the general purpose computation abilities of modern GPUs, in addition to their rendering abilities, to be able to localise at approximately 2 Hz.

An alternative to localisation is known as visual servoing [9], wherein images are used to directly build a control law for following a path previously traversed by a robot. The prior or map in this case simply consists of a series of key images from the original path traversal. In [10], this approach is demonstrated with the use of mutual information to determine the similarity between images. This approach is extremely dependent on the viewpoint of the robot, and limits the robot to tracking the exact same path as was recorded previously. In contrast, using a pointcloud or mesh as a prior allows synthetic images to be generated from any viewpoint, leading to more flexibility in the paths a robot can follow and the configuration of the sensors.

Dense tracking and mapping (DTAM) [4] is another method that uses a dense texture as a prior, which is built in real-time in parallel with motion tracking. In this approach, average photometric error is used as a cost function — every pixel in the live image and the projection of the prior is compared individually, and the average of the difference taken. Computation of this cost function is highly data parallel, and real-time performance on a GPU is achieved. Because the method depends on the actual pixel values, however, DTAM is not robust under appearance changes such as variation in illumination or use of a different camera. For outdoor scenes, where significant short-term changes in appearance occur throughout the day, in addition to gradual longer-term changes, this approach is not suitable. We address this by using a whole-image information theoretic measure of image similarity, which depends only on the distribution of pixel values, not the absolute values.

A different approach is demonstrated by Seq-SLAM [11], wherein sequences of images are employed. The use of downscaling, patch normalisation, and cross-correlation are shown to produce robustness to extreme changes in appearance. This method, however, is not capable of precise metric localisation, being limited to topological place recognition. Furthermore, it has been shown by [12] to be highly dependent on viewpoint.

Finally, a mesh coloured by laser reflectance rather than visual appearance is used as a prior for camera-based localisation by [13]. The authors note that this produces a prior unaffected by lighting conditions at the time at which it was captured. Furthermore, only the ground plane is included in the prior, as inclusion of the rest of the scene doubled the evaluation time. No gradient-based optimisation is carried out — instead, a grid search over a small 3-degree-of-freedom (DoF) region (2 translational dimensions, 1 rotational) is run to find the pose which minimises Normalised Mutual Information[1]. Root-mean-square (RMS) errors of 14 to 45 cm are reported, and localisation can be performed at a rate of approximately 10 Hz [13]. We wish to pursue full 6-DoF localisation, however, making the use of a grid search as in this paper intractable.

III. Mapping And Mesh Synthesis

Our approach to image-based localisation relies on having available a high-quality prior representation of the scene. For this system, we have chosen to use a 3-dimensional textured mesh as our prior. This provides richer information about the scene than simply using a pointcloud, as shown by [4].

Our mapping approach consists of two stages: first we generate the geometry of the mesh using LIDAR data, then we texture it using camera images.

A. Meshing

In order to build our mesh, we use a vehicle equipped with a 2-D push-broom LIDAR scanner and a number of cameras. We use visual odometry from a forward-facing stereo camera to estimate the motion of the vehicle between subsequent scans, giving us the 3-dimensional location of each scan. We then create our mesh by simply stitching together adjacent points from neighbouring scans, discarding anomalous triangles. Whilst the visual odometry does not provide us with globally accurate maps, it is good enough to create locally coherent metric maps over short distances. Figure 2(a) shows the untextured mesh.

We store the map as a series of 'slices,' each consisting of the triangles between two adjacent scans. This allows us to easily dynamically load any subsection of the map that we wish with minimal overhead. Most of the triangles in the mesh are on the order of a few centimetres in side length — we discard triangles with sides greater than 1 metre in length to avoid creating anomalous surfaces.

B. Texturing

To generate textures for our map, we project the mesh into each image from the survey vehicle's camera in turn.

[1]Normalised Mutual Information is a simple transform of Normalised Information Distance

(a) (b) (c)

Fig. 2. Generation of the prior: (a) First we build a mesh by stitching together subsequent scans from the push-broom LIDAR; (b) we then project the mesh in turn into each camera image, and generate a texture for each triangle — here we show all the textures for a single 'slice,' from many different input images; and (c) we combine all the slices with their textures for the final prior.

This camera does not need to be the same one used for localisation. For each triangle, we choose the image in which its projection is largest — this gives us the highest resolution textures. Whilst this does have a tendency to choose triangles at the edges of distorted images, introducing a preference for images in which a triangle projected to the centre of the image did not give a noticeable improvement in the quality of the resulting textures. Figure 2(b) shows all the textures for the triangles in a single slice, whilst Figure 2(c) shows the final result.

IV. FARLAP

In this section we describe our proposed localisation system. Our system is based on finding the pose at which a synthetic image produced from the prior best matches the live camera image. We first describe the metric we use to determine how well two images match, then describe how we carry out the optimisation.

A. Normalised Information Distance

In order to choose the pose which best matches the prior to live camera images, we require a function for evaluating image similarity. We cannot simply take the differences between respective pixels as in [4], as due to the reliance on pixel values this method is not robust to changes in appearance. Given that road vehicles need to be able to localise under many different lighting conditions and long term changes in appearance, this is not acceptable. Furthermore, different cameras will produce different colours, and we do not want to be limited to only being able to localise with the same camera as was used to build the map.

Mutual information is a common statistic used for image registration, as it provides a robust way to compare entire images [14]. The fact that mutual information depends only on the distribution of pixel values, not the actual values, means that it is usable with images captured by different sensors, and is robust to changes in appearance. Mutual information, however, has the problem of being dependent on how much overlap there is between two images [13] — this causes problems for us, as our synthetic priors will have large gaps (e.g. sky). We instead choose to use Normalised Information Distance (NID) [15], a metric with similar properties to mutual information, but which is not dependent on the number of pixels of overlap.

Given two images, A and B, the NID is given by (1).

$$NID(A,B) = \frac{H(A,B) - MI(A,B)}{H(A,B)} \tag{1}$$

Here, H and MI represent entropy and mutual information respectively, given by (3) and (4).

$$H(A) = -\sum_{a \in A} p_a \log(p_a) \tag{2}$$

$$H(A,B) = -\sum_{a \in A}\sum_{b \in B} p_{ab} \log(p_{ab}) \tag{3}$$

$$MI(A,B) = H(A) + H(B) - H(A,B) \tag{4}$$

where p_a and p_{ab} are probabilities of binned intensity values from an image histogram and joint histogram respectively. We use histograms with 32 bins.

B. Image Distortion

The cameras we use for localisation have very wide-angle lenses. Attempting to undistort these images results in large regions of the image being interpolated from a small number of pixels. Rather than undistorting live images, we thus chose to distort the synthetic image generated from the prior. As we can distort the mesh *before* texturing it, we are able to get sharp textures in the distorted image. The distortion can be done on the GPU in the same vertex shader that carries out the original projection — we never actually need to generate the undistorted projection of the mesh. As a result, applying distortion adds almost no time to the rendering pipeline. Furthermore, building these images only requires a discrete lookup table obtainable by intrinsic calibration — no knowledge of the actual camera distortion function is needed. We obtain these models using the OCamCalib toolbox [16].

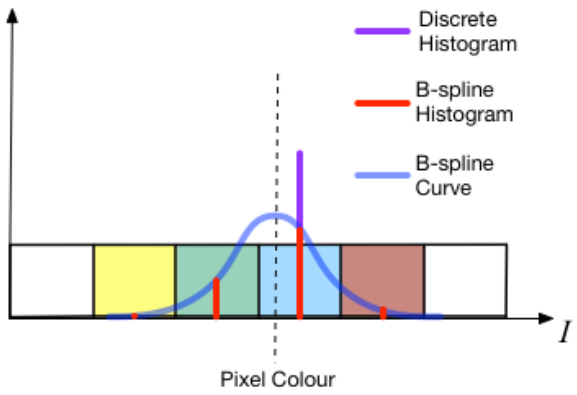


Fig. 3. Building a differentiable histogram. Rather than incrementing only the bin into which a pixel value falls, we add the coefficients from a cubic B-spline using supports at the centre of each bin. Because the histogram values now depend not only on which bin a pixel falls into, but *where* in that bin the pixel falls, the histogram becomes a continuous function of the pixel values. This diagram shows the effect of adding a single pixel to the histogram.

C. Optimisation

Given a 6-DOF vehicle pose relative to our prior map, G_{VP}, we can generate a virtual image $I^*(G_{VP})$ showing what a camera should be seeing at that point. We wish to find the pose $\hat{G}_{VP}$ at which the NID between this synthetic image and the live camera image I is minimised.

$$\hat{G}_{VP} = \min_{G_{VP}} NID(I, I^*(G_{VP})) \tag{5}$$

We use a grayscale intensity for each pixel as our signal.

We use the quasi-Newton Broyden-Fletcher-Goldfarb-Shanno (BFGS) algorithm [17] implemented in Ceres Solver [18] which simultaneously estimates the Hessian and carries out the minimisation. Usage of BFGS requires us to calculate the analytical Jacobian of the NID (cost function) with respect to the the 6-DOF pose (the function inputs). The derivative of NID with respect to the vehicle pose is as follows:

$$\frac{dNID}{dG_{VP}} = \frac{\frac{dH(I,I^*)}{d\mathbf{G}_{VP}} MI(I,I^*) - H(I,I^*)\frac{dMI(I,I^*)}{d\mathbf{G}_{VP}}}{H(I,I^*)^2} \tag{6}$$

The entropy derivatives are given by (7) and (8), where n is the number of histogram bins.

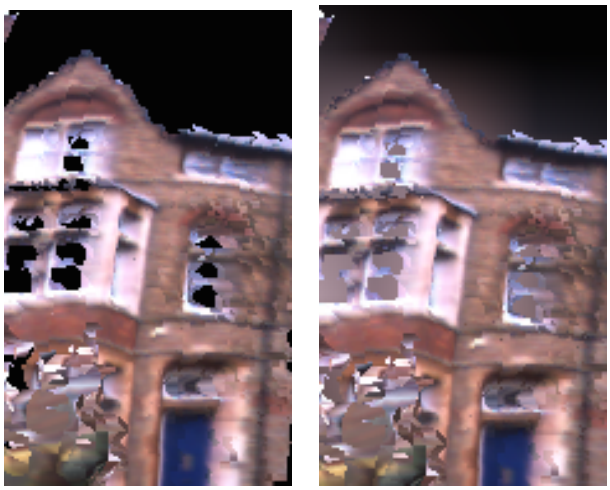
$$\frac{dH(I)}{dG_{VP}} = -\sum_{b=1}^{n} \frac{dp_b}{dG_{VP}}(1 + \log p_b) \tag{7}$$

$$\frac{dH(I,I^*)}{dG_{VP}} = -\sum_{a=1}^{n}\sum_{b=1}^{n} \frac{dp_{ab}}{dG_{VP}}(1 + \log p_{ab}) \tag{8}$$

In a standard formulation of a histogram, the values in the bins are discrete, and thus not differentiable. We therefore have to alter our histogram construction to get a continuous function for the histogram distribution. Rather than incrementing the bin into which the intensity of each pixel falls, we instead use cubic B-Spline interpolation, as in [5], to update four bins for each pixel. This is demonstrated in Figure 3. As we are using a cubic B-spline, these coefficients are all twice differentiable, thus we can now compute analytic derivatives of the histogram values with respect to changes in pixel appearance.

To find the Jacobian of pixel intensities with respect to screen coordinates, we once again make use of a cubic B-spline interpolation to get a continuous, differentiable intensity surface. We run a B-spline pre-filter over the synthetic image generated from the prior, as described in [19]. This gives us a differentiable image with pixel values, which when used as the control points for a B-spline, yield the original values at the pixel centres. We can then recover the required Jacobian by differentiating the B-spline surface.

The synthetic image we generate from the map has large gaps in it. This is due to the fact that some areas (e.g. the sky) are not meshed. We ignore these regions in our NID calculations, but if we simply built our B-spline surface over this raw image, our derivatives would be dominated by the sharp changes at the edges of the non-textured regions. Instead, we render our synthetic image at several resolutions — OpenGL can do this in hardware, with negligible overhead — and use lower resolution images to fill the gaps in the higher resolution images. An example of this is shown in Figure 4. This, in effect, blurs the edges of the non-textured regions, thus preventing large derivatives at those boundaries. Note that the gaps in the original (full resolution) image are still ignored in the NID calculation — the smoothing is simply used to prevent sharp derivatives at the boundaries.



(a) (b)

Fig. 4. We render our synthetic images at two resolutions, and use the lower resolution version to fill in gaps in the higher resolution image. This prevents sharp changes at the edges of the mesh from dominating image derivatives. (a) A segment of the higher resolution image. (b) The same image filled in with a slightly lower resolution image, producing blurring around edges and filled-in holes in the windows.

Similarly, our camera distortions are specified in the form of a discrete lookup table which maps undistorted image coordinates to distorted image coordinates. As we require a differentiable function, we once again use B-spline interpo-

lation.

The final Jacobian we require is that which maps derivatives in the vehicle frame to those in image space. Here we make use of the depth buffer available in the OpenGL rendering pipeline. The Jacobian is given by (9) [7].

$$J = \begin{bmatrix} -1/z & 0 & u/z & uv & -(1+u^2) & v \\ 0 & -1/z & v/z & 1+v^2 & -uv & -u \end{bmatrix} \quad (9)$$

where u and v are the horizontal and vertical image coordinates respectively, and z is the depth value. As this Jacobian operates on each pixel independently, requiring only the pixel coordinates, it can be computed during rendering in the OpenGL shader pipeline. In addition to the actual textured image, we render four derivative images. Each of the twelve derivatives (two image coordinates with respect to each of six degrees of freedom) is rendered to a channel of one of these images. Figure 5 shows an example of derivatives in image form.

At each step of the BFGS optimisation, we render a new scene at the current pose, and calculate the NID and Jacobians based on that scene. This results in every evaluation being based on the prior being sampled at exactly the resolution of the live camera image — the prior is sampled at the center of every pixel. As a result, our method is not provably convergent, as we are effectively changing the data used for each evaluation — a new synthetic image is used each time. We demonstrate, however, that in practice the function has a large convergence basin.

The entire cost function pipeline is evaluated on the GPU. Synthetic camera images from the prior are generated by OpenGL, and shared with OpenCL [20] kernels which carry out the histogram and NID calculations. As a result, not only do we exploit the data-parallel nature of our computations, but we also avoid the time-consuming reading of entire images back from the GPU. This allows us to localise at a rate of approximately 2 Hz.

V. Experimental Setup

In this section we describe the approach we use to evaluate and demonstrate the efficacy of our localisation system.

The platform we use to build our priors and localise is a Bowler Wildcat. We use a vertical SICK LMS-151 laser scanner mounted at the front of the vehicle to generate our maps, with a Point Grey Bumblebee2 mounted at the front of the vehicle used to obtain visual odometry. Our meshes are coloured using a Point Grey Ladybug2 spherical camera.

We have four Point Grey Grasshopper2 monocular cameras, fitted with wide-angle fisheye lenses, for use in localisation. These cameras are mounted pointing slightly down, approximately 45°left and right of the forwards and rear direction of the vehicle respectively.

We carry out our evaluations on a computer with a 2.4 GHz Intel Xeon processer, 6 GB of RAM and an NVIDIA GeForce GTX Titan GPU, running Ubuntu 12.04.

VI. Results

To assess the ability of our system to localise in a real-world environment, we drove eight laps around a loop of public roads in north Oxford (see Fig. 6), totalling approximately 5.6 km. We built a mesh prior from the laser and Ladybug2 data. We then localised using the same dataset, but with Grasshopper2 cameras. This allows us to have a very accurate ground truth for comparison, as we are using the same odometry feed as was used to build the map. Although the structure of the scene is thus the same during localisation as during the map-building phase, given that we are using different images from those used to texture the map, there are large differences in appearance.

A. Localisation Performance

TABLE I
RMS error in each degree of freedom.

	RMS error
x (m)	0.0742
y (m)	0.0373
z (m)	0.0490
r_x (°)	0.9183
r_y (°)	0.3159
r_z (°)	0.3571

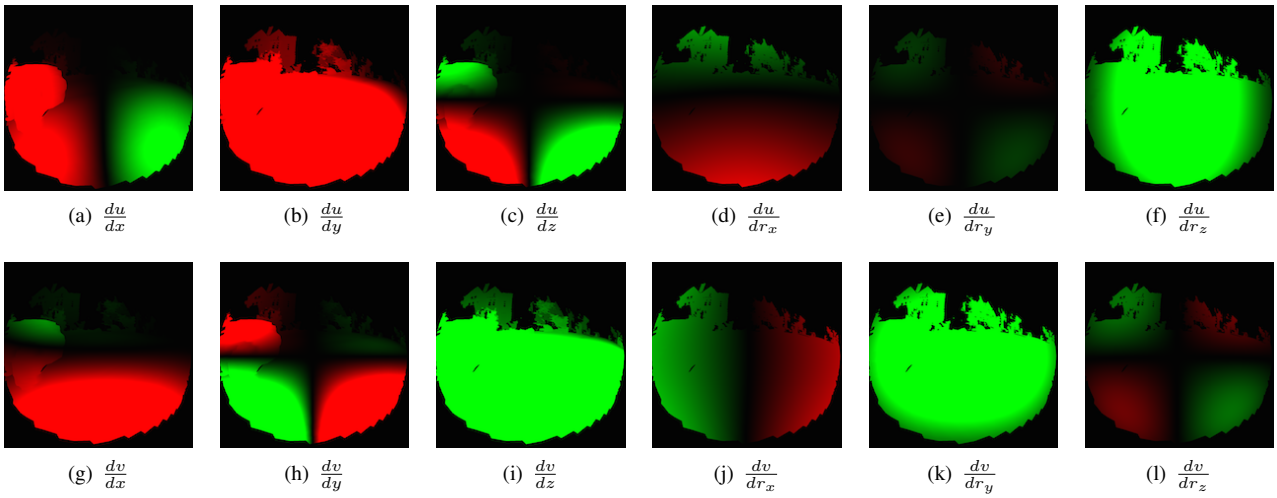
Figure 7 shows the errors between the localised solution and the ground truth in each degree of freedom. Approximately 7% of locations failed to converge on a solution — these are not shown in the histograms. The vast majority of locations show an error of less than 5 cm in the translational dimensions, and less than 1° in the rotational dimensions. Table I shows the RMS errors for each of the axes.

In Figure 8, we show how the NID varies with movement in each axis from the ground truth position. The functions all have a minimum indistinguishable from the ground truth position, with the exception of pitch, whose minimum is less than a degree from zero. Note that the curve from z has a local minimum at approximately $+1.5$ m — this is due the virtual camera being below the road, and the projection of the road looking very similar from this position as it does from above.

B. Time Performance

In Figure 9(a), we show a histogram of the time taken to evaluate the NID between the live camera image and our prior. The mean time taken to evaluate the cost function is approximately 35 ms, but the majority of evaluations take less than 25 ms. The vast majority of the time taken in the NID evaluation is used in building the joint histogram between the two images — this requires essentially random memory access for writing to the histogram, resulting in less of a speed advantage on the GPU compared to other parts of the pipeline.

A single localisation generally requires approximately 20 cost function evaluations, as can be seen in Figure 9(b). Over the course of our 6km of data, localisation can be performed at approximately 2 Hz.



(a) $\frac{du}{dx}$ (b) $\frac{du}{dy}$ (c) $\frac{du}{dz}$ (d) $\frac{du}{dr_x}$ (e) $\frac{du}{dr_y}$ (f) $\frac{du}{dr_z}$

(g) $\frac{dv}{dx}$ (h) $\frac{dv}{dy}$ (i) $\frac{dv}{dz}$ (j) $\frac{dv}{dr_x}$ (k) $\frac{dv}{dr_y}$ (l) $\frac{dv}{dr_z}$

Fig. 5. Derivative images generated during rendering. These show how the image coordinates of a point change with changes in camera pose. u and v are horizontal and vertical image coordinates respectively. Green indicates a positive derivative, red indicates negative. We use a frame with x into the screen, y to the right of screen, and z pointing down, and r_x, r_y and r_z represent rotations about the x, y and z axes respectively. Note that for visualisation purposes, the derivatives use two channels to display positive and negative values respectively — in the localisation pipeline only a ssingle channel is required.

Fig. 6. Experimental test site in north Oxford.

C. Effect of Field of View

Here we evaluate the effect on localisation performance of using a camera with a wide-angle fisheye lense. The OCamCalib [16] toolbox we use for generating camera models allows us to build models for different fields of view, with wider fields of view yielding more distorted images. Figure 10 shows NID curves averaged across the three translational and rotational axes respectively.

Up to around a 120° field of view, increasing the field of view increases the smoothness of the cost function, both making the convergence basin deeper and the minimum more accurate. This allows us to localise accurately with an initial estimate as much as 1.3 m away from the true position in horizontal directions, and 10° in rotation.

When the field of view is increased beyond 120°, the minimum of the cost functions start to become less clear. At 148°, localisation is still possible — although the rotational minima are not precise, they are still within 5° of the true value. The loss in precision, however, brings a wider convergence basin. Beyond 150°, the cost functions become virtually unusable.

With extremely large fields of view, small changes in the pose of the camera do not change the image much, resulting in much flatter cost functions. At extremely narrow fields of view, only a small part of the camera image is being used, and as a result not much information is available for localisation. The 120°field of view result gives the best combination of a wide convergence basin and accurate minimum.

VII. Conclusions

We have presented a new system that can successfully localise a vehicle within a textured prior using only a monocular camera. We demonstrated the performance of our system on 6 km data captured on real-world roadways, showing it to be both robust and precise, with RMS errors of less than 8 cm and 1 degree and a wide convergence basin of over 1.3 metres in each direction. Due to the data-parallel nature of the problem, we are able to exploit GPU computation to carry out localisation at approximately 2 Hz. The ability to robustly localise with centimetre-level precision using only monocular cameras is a strong step towards enabling low-cost autonomy for road vehicles.

VIII. Acknowledgements

The authors wish to acknowledge the following funding sources. Geoffrey Pascoe is supported by funding from the Rhodes Trust. Will Maddern and Alex Stewart are supported by EPSRC grants, and Paul Newman is supported by EPSRC Leadership Fellowship EP/J012017/1.

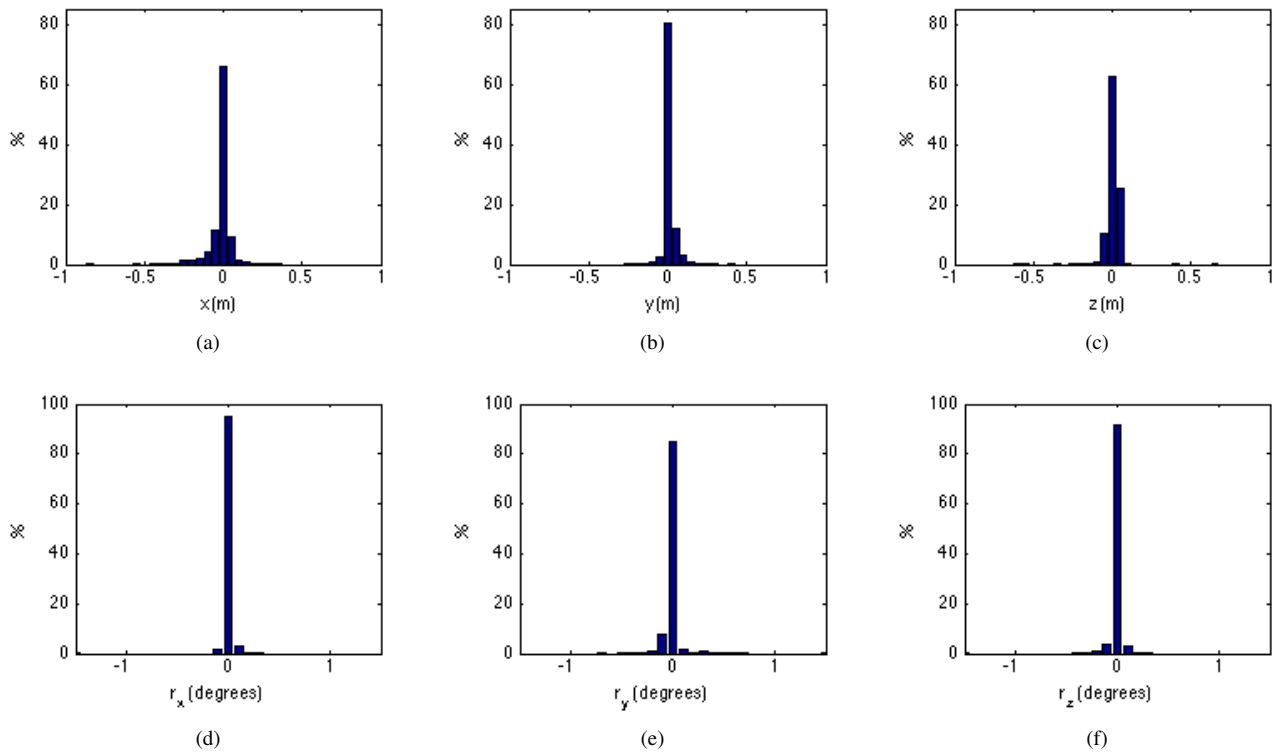


Fig. 7. Histograms showing displacement between the localised solution and ground truth in each degree of freedom. The coordinate system places x in the direction of the vehicle's travel, y to the vehicle's right, and z downwards into the road.

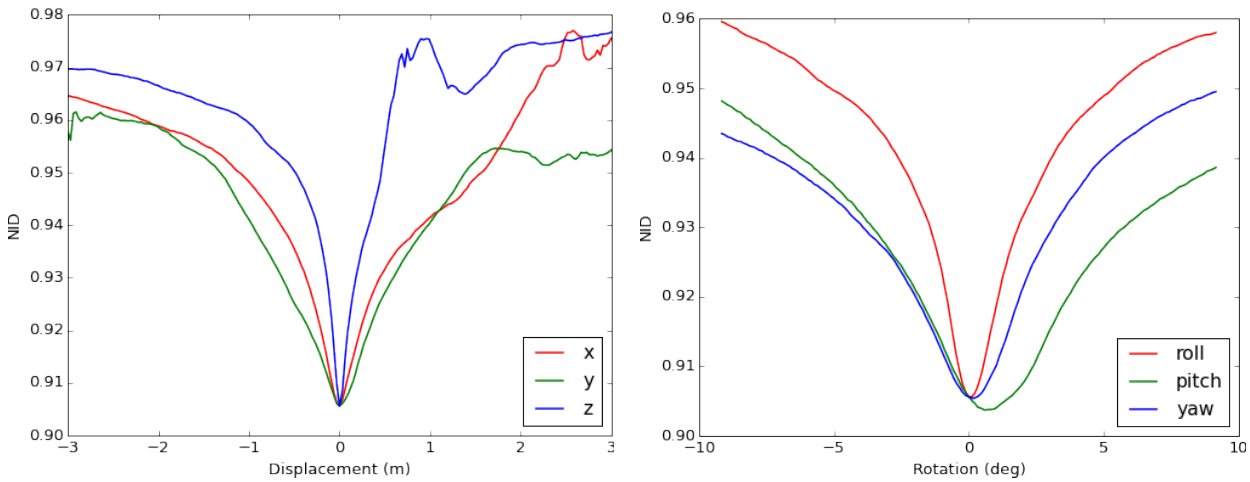


Fig. 8. NID values at varying offsets from the ground truth pose. These plots are averaged across 20 different locations. (a) Translational axes with x, y and z shown as red, green and blue respectively. (b) Rotational axes with roll, pitch and yaw shown as red, green and blue respectively. The second local minimum in the z axis is due to the road looking very similar when rendered from below as it does from above.

REFERENCES

[1] J. Levinson, M. Montemerlo, and S. Thrun, "Map-Based Precision Vehicle Localization in Urban Environments," *Proc. of Robotics: Science and Systems (RSS)*, 2007.

[2] J. Levinson and S. Thrun, "Robust vehicle localization in urban environments using probabilistic maps," *2010 IEEE International Conference on Robotics and Automation*, pp. 4372–4378, May 2010.

[3] Velodyne, "A High Definition Lidar Sensor for 3-D Applications," 2007. [Online]. Available: http://velodynelidar.com/lidar/products/white_paper/HDL%20white%20paper_OCT2007_web.pdf

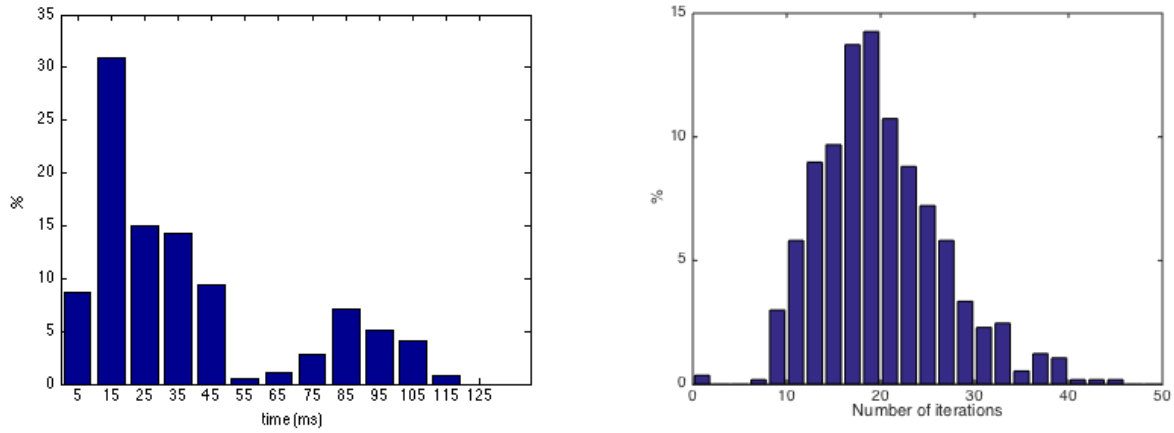
[4] R. A. Newcombe, S. J. Lovegrove, and A. J. Davison, "DTAM: Dense tracking and mapping in real-time," *Computer Vision (ICCV), 2011 IEEE International Conference on*, pp. 2320–2327, Nov. 2011.

[5] A. D. Stewart and P. Newman, *LAPS - localisation using appearance of prior structure: 6-DoF monocular camera localisation using prior pointclouds*. IEEE, 2012, pp. 2625–2632.

[6] W. Maddern, A. D. Stewart, and P. Newman, "LAPS-II: 6-DoF Day and Night Visual Localisation with Prior 3D Structure for Autonomous Road Vehicles," in *Intelligent Vehicles Symposium, 2014 IEEE*, 2014.

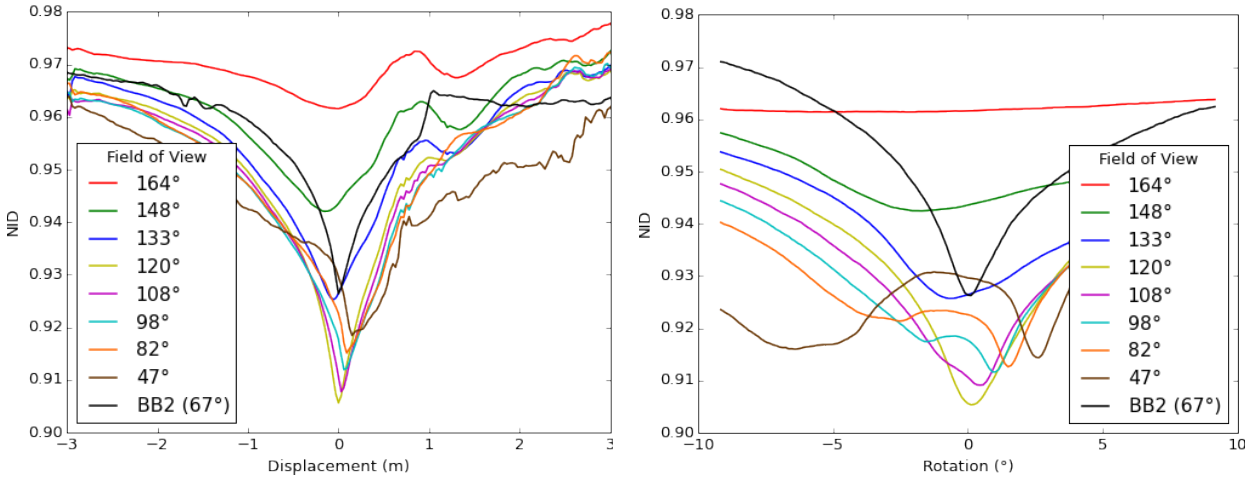
[7] G. Caron, A. Dame, and E. Marchand, "Direct model based visual tracking and pose estimation using mutual information," *Image and Vision Computing*, vol. 32, no. 1, pp. 54–63, Jan. 2014.

[8] Khronos, "OpenGL - The Industry's Foundation for High Performance



(a) (b)

Fig. 9. Runtime behaviour of optimiser. (a) Time taken to evaluate the NID function, including rendering the synthetic image. The majority of cost function evaluations take less than 25 ms. (b) Number of cost function evaluations carried out in each optimisation. Most optimisations require approximately 20 evaluations.



(a) (b)

Fig. 10. NID curves with varying fields of view. Each curve is the mean of the three translational or rotational axes, averaged across 20 locations.

Graphics," accessed 9/8/2014. [Online]. Available: http://www.khronos.org/opengl/

[9] S. Hutchinson, G. Hager, and P. I. Corke, "A tutorial on visual servo control," *IEEE International Conference on Robotics and Automation*, vol. 12, pp. 651–670, 1996.

[10] A. Dame and E. Marchand, "Using mutual information for appearance-based visual path following," *Robotics and Autonomous Systems*, pp. 1–32, 2012.

[11] M. J. Milford and G. F. Wyeth, "SeqSLAM: Visual route-based navigation for sunny summer days and stormy winter nights," in *Proceedings - IEEE International Conference on Robotics and Automation*, 2012, pp. 1643–1649.

[12] N. Sünderhauf, P. Neubert, and P. Protzel, "Are we there yet? Challenging SEQslam on a 3000 km journey across all four seasons," in *Proc. of Workshop on Long-Term Autonomy, IEEE International Conference on Robotics and Automation (ICRA)*, 2013, p. 2013.

[13] R. W. Wolcott and R. M. Eustice, "Visual Localization within LIDAR Maps for Automated Urban Driving," in *Intelligent Robots and Systems (IROS), IEEE/RSJ International Conference on*, 2014.

[14] J. P. W. Pluim, J. B. A. Maintz, and M. a. Viergever, "Mutual-information-based registration of medical images: a survey." *IEEE transactions on medical imaging*, vol. 22, no. 8, pp. 986–1004, Aug. 2003.

[15] M. Li, X. Chen, X. Li, B. Ma, and P. M. B. Vitanyi, "The similarity metric," *Information Theory, IEEE Transactions on*, vol. 50, no. 12, pp. 3250–3264, 2004.

[16] D. Scaramuzza, A. Martinelli, and R. Siegwart, "A toolbox for easily calibrating omnidirectional cameras," in *IEEE International Conference on Intelligent Robots and Systems*, 2006, pp. 5695–5701.

[17] J. Nocedal and S. J. Wright, *Numerical Optimization*, 1999, vol. 43.

[18] S. Agarwal, K. Mierle, and Others, "Ceres Solver." [Online]. Available: http://ceres-solver.org

[19] D. Ruijters and P. Thévenaz, "GPU prefilter for accurate cubic B-spline interpolation," *The Computer Journal*, vol. 55, pp. 15–20, 2012.

[20] Khronos, "OpenCL - The open standard for parallel programming of heterogeneous systems," accessed 9/8/2014. [Online]. Available: http://www.khronos.org/opencl/