Super-Resolved Texture Tracking of Planar Surface Patches

Frank Dellaert Chuck Thorpe Sebastian Thrun
Computer Science Department and The Robotics Institute
Carnegie Mellon University, Pittsburgh PA 15213

Abstract

We present a novel approach to tracking planar surface patches over time. In addition to tracking a patch with full six degrees of freedom, the algorithm also produces a super-resolved estimate of the texture present on the patch. This texture estimate is kept as an explicit model texture image which is refined over time. We then use it to infer the 3D motion of the patch from the image sequence.

The main idea behind the approach is to use a technique from computer graphics, known as texture mapping, as the measurement model in an extended Kalman filter. We also calculate the partial derivative of this image formation process with respect to the 3D pose of the patch, which functions as the measurement Jacobian. The super-resolved estimate of the texture is obtained using the standard extended Kalman filter measurement update, with one essential approximation that makes this computationally feasible. The resulting equations are remarkably simple, yet lead to estimates that are properly super-resolved.

In addition to developing the theory behind the approach, we also demonstrate both the tracking and the super-resolution aspect of the algorithm on real image sequences.

1 Introduction

In this paper we present a novel approach to track *planar surface patches* in a sequence of images. In addition to an estimate of the 3D pose parameters over time (position and orientation), our algorithm produces a super-resolved estimate of the texture information that is present on the patch. This texture estimate is kept as an explicit model texture image which is refined over time. We then use it to infer the 3D motion of the patch from the image sequence. In practice, this is done using an extended Kalman filter, which is also used to update and refine the texture estimate over time.

The fact that a texture estimate is obtained during tracking makes the approach applicable in a number of domains, where tracking and resolving texture are both important, for example modeling for virtual reality. In addition, if the texture estimate is kept at a higher resolution than the original image sequence, we can *super-resolve* the texture on the surface. This opens the door for new applications where achieving super-resolution while tracking is useful. For example, one could perform optical character recognition on the texture estimated from some moving planar surface. One example application is to to read the license plate attached to a moving car, or read traffic signs that are being tracked from a moving vehicle.

We know of one other instance in which a texture is explicitly estimated for the purpose of tracking. This is the work by La Cascia, Isodoro and Sclaroff [5] on head tracking . They use mosaicing techniques to update a texture map, which is then used to track a person's head. They do this by registering incoming video images to the texture, mapped onto a shape model of the head. Our approach is similar in idea, but very different in its formulation. We use a Bayesian estimation framework in the form of the Kalman filter. This allows us to reason about confidence in the texture map in a more principled way. It is this firm grounding in Bayesian estimation theory that makes it possible to achieve super-resolved texture estimates, something which is not attempted in [5].

Cheeseman et al. [1] is the most referenced work on super-resolution in the field of computer vision. Like us, they use a Bayesian approach to registering multiple images of a surface. Their goals and assumptions are significantly different however. Whereas we are concerned with the *on-line* tracking of a textured surface in a video stream, they attempt to combine images off-line, using an iterative minimization to solve for registration between images and the construction of a super-resolved estimate. There also exists a large body of work in the video processing literature that deals with the super-resolved reconstruction of video-imagery. Two good overviews of recent work are by

Schultz and Stevenson [9], and by Patti et al [7]. However, this work is concerned only with 2D images; no attempt is made to model motion of objects in 3D.

2 Approach

The main idea of our approach lies in the use of *texture mapping*, a technique from computer graphics, to predict the appearance of a textured surface patch in the image. In addition, we also calculate the *derivative* of this image formation process with respect to the position and orientation of the patch. We then use both the model and its derivative in an extended Kalman filter to track the patch over time in a sequence of images. Finally, we also use the derivative to recursively update a model of the texture on the surface being tracked.

The state variables we are trying to estimate are the three-dimensional position, orientation, and the texture of a patch. To specify the 3D pose, we identify a planar patch with a *local coordinate frame* L, situated in a global *world coordinate frame* W (as in [10]). We align the surface normal with the local Z axis, so that the planar surface is defined by the X and Y axes. Thus, the position of the planar patch is defined by the six-dimensional state vector $\mathsf{x}_p = [X\,Y\,Z\,\psi\,\theta\,\phi]^T$, where $[X\,Y\,Z]^T$ is the origin of the local frame L, and $[\psi\,\theta\,\phi]^T$ are a set of Euler angles specifying its orientation with respect to the world coordinate frame W.

A defining characteristic of our approach is that we also estimate a model $\hat{\mathcal{T}}$ of the texture $\mathcal{T}$ that is present on the patch. This model is stored as an array of estimated intensity values, or alternatively, RGB values. Following Cheeseman's terminology [1], we will call these elements *mixels* (for *m*odel p*ixels*). The total state x to be estimated is then the tuple (x_p, $\mathcal{T}$), i.e., the 3D pose and the texture.

To estimate these variables over time we use an extended Kalman filter. A Kalman filter is a recursive measurement processing algorithm, that under certain assumptions optimally tracks the state of a system [6]. The Kalman filter takes a Bayesian approach to the estimation problem, representing the uncertainty about the state as a probability density over states. As a consequence of the assumptions made, this probability density is a multivariate Gaussian, and remains Gaussian at all times. Thus, we define the *state estimate* $\hat{\mathsf{x}}(t)$ and the *state covariance matrix* $\mathsf{P}(t)$, together completely specifying a Gaussian density that encodes our knowledge about the true state $\mathsf{x}(t)$. In our case, we use an *extended* Kalman filter, which approximates the optimal linear filter in the case of non-linear dynamics and measurements.

In the sections below we explain our approach by detailing how we implement each step of the main Kalman filter loop. Specifically, our algorithm continually loop over four steps:

1. Predicting Patch Position (Section 2.1): predict *where* we will find the patch in the next image.

2. Predicting Patch Appearance (Section 2.2): predict *what* the patch will look like in the image.

3. Updating the Position Estimate (Section 2.3): use the difference between predicted and actual image to refine our patch *position* estimate $\hat{\mathsf{x}}_p$.

4. Updating the Texture Estimate (Section 2.4): use the difference between predicted and actual image to refine our *texture* estimate $\hat{\mathcal{T}}$.

These steps implement the dynamics update, measurement prediction and measurement update steps of the Kalman filter, respectively. Our contribution lies in the specific form of the measurement model that is used, and in calculating its Jacobian.

2.1 Predicting Patch Position

The *dynamics update* step serves to predict the new pose $\hat{\mathsf{x}}_p$ of the patch, given our estimate at the previous time step and a motion model. We simply use the standard extended Kalman filter formulation, reviewed briefly below. One important approximation we make is neglecting the cross-correlation terms between texture and position variables, i.e., in this dynamics update step we act as if the position portion of the state x_p is independent of the texture estimate $\hat{\mathcal{T}}$, and we only update that portion of the state here.

Specifically, we assume that a model for the *dynamics* of the patch is available in the form of a continuous stochastic differential equation:

$$\dot{\mathsf{x}}_p(t) = \mathsf{f}[\mathsf{x}_p(t), \mathsf{u}(t), t] + \mathsf{G}(t)\mathsf{w}(t) \qquad (1)$$

where $\mathsf{f}(.,.,t)$ encodes the possibly non-linear dynamics, $\mathsf{u}(t)$ represents deterministic control inputs, and $\mathsf{G}(t)$ distributes the Gaussian white noise process $\mathsf{w}(t)$ over the variables of interest [6]. Although in general f, G, and w are time-variant, we will omit their time dependency below for clarity's sake.

The pose estimate ($\hat{\mathsf{x}}_p(t)$, P_p) is propagated in time by integrating the dynamics equations forward to the

current time, starting from the previous estimate [6]:

$$\dot{\hat{\mathsf{x}}}_p(t) = \mathsf{f}[\hat{\mathsf{x}}_p(t), \mathsf{u}(t)] \quad (2)$$

$$\dot{\mathsf{P}}_p(t) = \mathsf{F}(t)\,\mathsf{P}_p(t) + \mathsf{P}_p(t)\,\mathsf{F}^T(t) + \mathsf{G}\,\mathsf{Q}\,\mathsf{G}^T \quad (3)$$

where $\mathsf{F}(t)$ is the Jacobian of the system dynamics f, and Q is the covariance kernel of the white noise process w. $\mathsf{F}(t)$ represents the linearization of f around the estimate $\hat{\mathsf{x}}_p(t)$ and will in general vary with time, *even if* f *is time-invariant.*

In many practical applications a model for the system dynamics will be available. E.g., when tracking objects from a moving vehicle, the vehicle dynamics can be accurately modeled, and the movement of the objects to be tracked is related to the vehicle movement [2]. In the worst case, when no domain knowledge is available, one can simply use a constant velocity model [6], with appropriate noise terms to compensate for the uncertainty in the model dynamics.

2.2 Predicting Patch Appearance

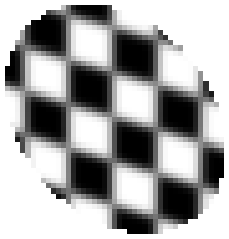

Figure 1: Example of a disc-shaped planar patch with a checkerboard texture mapped onto it.

The main idea in our approach is to use *texture mapping* as the measurement model in the Kalman filter. Texture mapping is an image synthesis technique from computer graphics, in which a texture image is mapped onto a surface in a three-dimensional scene, much as wallpaper is applied to a wall [4, 8]. For example, Figure 1 shows the result of mapping a checkerboard pattern onto a circular patch, pitched 30 degrees forward, then rolled 30 degrees to the right.

Formally, we model the *measurement process* as a non-linear function h of the state, corrupted by a Gaussian white noise process $\mathsf{v}(t)$:

$$\mathsf{z}(t) = \mathsf{h}[\mathsf{x}(t), t] + \mathsf{v}(t) \quad (4)$$

In our case the *measurement* z is a subset of the current image, and consists of a collection of image intensity values $\mathcal{I}(\mathsf{p}, \mathsf{x}_p, \mathcal{T})$, one for each pixel p in the area occupied by image of the patch. As indicated, these pixel values will be a function of both the 3D pose x_p of the patch and its texture $\mathcal{T}$. Modeling the measurement can then be reformulated as asking the question: *given a pose estimate* $\hat{\mathsf{x}}_p$ *and a model of the texture* $\hat{\mathcal{T}}$*, what will be the value* $\mathcal{I}(\mathsf{p}, \hat{\mathsf{x}}_p, \hat{\mathcal{T}})$ *of each pixel in the image of the patch ?* This is exactly the problem addressed by texture mapping.

The simplest form of texture mapping is known as *point sampling*, and simply inverts the mapping m between (homogeneous) texture coordinates (s, t, u) and image coordinates (x, y, w). In this scheme, each pixel is inverse-mapped to its *pre-image* in texture space, and assigned the value of the nearest integer texture coordinate [8]:

$$\mathcal{I}(\mathsf{p}, \hat{\mathsf{x}}_p, \hat{\mathcal{T}}) = \hat{\mathcal{T}}(round(\mathsf{m}^{-1}(\mathsf{p}))) \quad (5)$$

In our case, the dependence on the pose estimate $\hat{\mathsf{x}}_p$ is subsumed in the mapping m. If we define a 4×4 matrix function ${}^{\mathsf{w}}_{|}\mathsf{T}(\mathsf{x}_p)$ to be the homogeneous coordinate transformation between local patch coordinates and world coordinates, collect the camera parameters in a 3×4 ${}^{|}_{\mathsf{w}}\mathsf{T}$, and define the scale of the texture in a 4×3 coordinate transformation matrix ${}^{|}_{\mathsf{t}}\mathsf{T}$, we obtain

$$\mathsf{p} = \mathsf{m}(\mathsf{k}) = {}^{|}_{\mathsf{t}}\mathsf{T}(\mathsf{x}_p)\,\mathsf{k} = {}^{|}_{\mathsf{w}}\mathsf{T}\,{}^{\mathsf{w}}_{|}\mathsf{T}(\mathsf{x}_p)\,{}^{|}_{\mathsf{t}}\mathsf{T}\,\mathsf{k} \quad (6)$$

i.e., a 3×3 matrix ${}^{|}_{\mathsf{t}}\mathsf{T}$, dependent on x_p, defining a *projective mapping* that maps points k in texture space to pixels p in image space.. Since this is a *2D to 2D* projective mapping, it is invertible, and we can apply the method outlined above.

To model the image formation process more realistically we use the *elliptical Gaussian filter*, a method due to Heckbert [4], for resampling the texture. Point sampling, as described above, leads to severe aliasing effects, i.e., it produces artifacts such as jaggies in the image. This can be remedied by convolving the texture with an appropriately shaped low-pass filter. We have chosen Heckbert's approach over alternative methods (such as bilinear interpolation) because (a) it is more realistic, (b) it is better at mapping high-resolution textures to lower-resolution images, and (c) the derivatives of the filters, which we will need below, are smooth and continuous. Thus, the pixel values are calculated by convolving the texture image $\hat{\mathcal{T}}$ with an *elliptical resampling filter* ρ centered around $\mathsf{m}^{-1}(\mathsf{p})$:

$$\mathcal{I}(\mathsf{p}, \hat{\mathsf{x}}_p, \hat{\mathcal{T}}) = \sum_{k \in \mathcal{Z}^2} \hat{\mathcal{T}}(\mathsf{k})\,\rho(\mathsf{m}^{-1}(\mathsf{p}), \mathsf{k}) \quad (7)$$

Lack of space prevents us from deriving a formula for ρ. However, the shape of the filter can be intuitively understood as follows: if you imagine a circular Gaussian filter in the image plane, and project it back

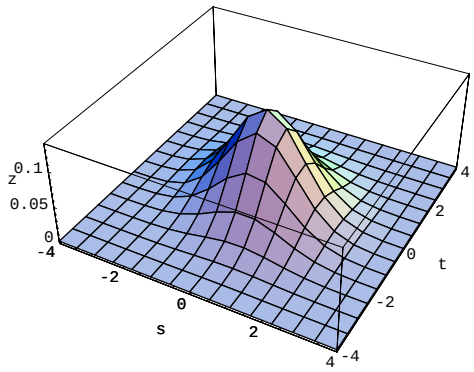

Figure 2: The elliptical resampling filter that was used to render the patch in Figure 1.

to the surface of the planar patch, the shape of this warped filter will depend on the pose of the patch. If it is parallel to the image plane, the projection will just be an enlarged circular Gaussian filter. If the patch is oriented arbitrarily, however, the projection will in general be an elliptical Gaussian filter. As an illustration, Figure 2 shows the resampling filter in texture space that was used to produce Figure 1. Note that it is rotated and elongated, as the patch is tilted with respect to the image plane.

2.3 Updating the Position Estimate

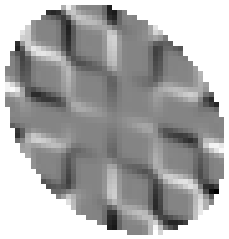

Figure 3: I_ψ: the partial derivative of Figure 1 with respect to yaw (rotating around Z). Gray is zero.

The key to making texture mapping work for the purposes of tracking, is to calculate the Jacobian of the measurement model h with respect to the pose state vector x_p. Indeed, most of the work in the Kalman filter is done in the *measurement update equations*[1]:

$$\mathsf{K} = \mathsf{P}_p\,\mathsf{H}^T\,[\mathsf{H}\,\mathsf{P}_p\,\mathsf{H}^T + \mathsf{R}]^{-1} \tag{8}$$

$$\hat{\mathsf{x}}_p \Leftarrow \hat{\mathsf{x}}_p + \mathsf{K}[\mathsf{z} - \mathsf{h}(\hat{\mathsf{x}}_p)] \tag{9}$$

$$\mathsf{P}_p \Leftarrow \mathsf{P}_p - \mathsf{K}\,\mathsf{H}\mathsf{P}_p \tag{10}$$

These equations express how the difference between the predicted measurement $\mathsf{h}(\hat{\mathsf{x}}_p)$ and the actual measurement z is used to calculate a new state estimate $\hat{\mathsf{x}}_p$. A crucial element in the computation of the gain matrix K, which determines how much weight will be given to the new measurement, is the *measurement Jacobian* H, defined as:

$$\mathsf{H}(\hat{\mathsf{x}}_p) \stackrel{\text{def}}{=} \left.\frac{\partial \mathsf{h}(\mathsf{x}_p)}{\partial \mathsf{x}_p}\right|_{\mathsf{x}_p = \hat{\mathsf{x}}_p} \tag{11}$$

In our case, we calculate the measurement Jacobian H *as the partial derivative of the texture mapping process with respect to the pose state variable* $\mathsf{x}_p = [X\,Y\,Z\,\psi\,\theta\,\phi]^T$. Thus, H will be composed of six partial derivatives, each expressing how the image of the patch will change in response to a small change in position or orientation. Interestingly, these partial derivatives can also be visualized as images. For example, in Figure 3, we have shown what happens when the patch is rotated around its surface normal. As is expected, most change occurs furthest away from the rotation center.

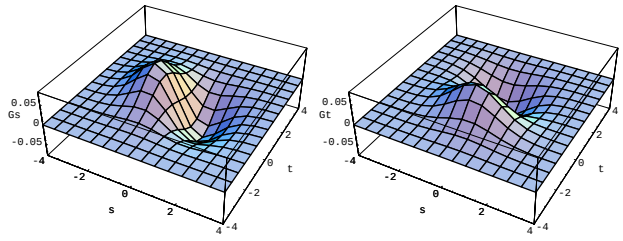

Figure 4: Gradient kernels

The way in which these *delta-images* are obtained provides considerable insight into the texture tracking process. If we rewrite ρ using separate, scalar functions $s(\mathsf{p})$ and $t(\mathsf{p})$ for the texture coordinates of $\mathsf{m}^{-1}(\mathsf{p})$, equation (7) becomes:

$$\mathcal{I}(\mathsf{p}, \hat{\mathcal{T}}) = \sum_{\mathsf{k} \in \mathcal{Z}^2} \hat{\mathcal{T}}(\mathsf{k})\,\rho(s(\mathsf{p}), t(\mathsf{p}), \mathsf{k}) \tag{12}$$

Taking the partial derivative of equation (12) with respect to (for example) yaw ψ, we get:

$$\begin{aligned}\frac{\partial \mathcal{I}(\mathsf{p}, \hat{\mathcal{T}})}{\partial \psi} =& \frac{\partial s(\mathsf{p})}{\partial \psi} \sum_{\mathsf{k} \in \mathcal{Z}^2} \hat{\mathcal{T}}(\mathsf{k})\,\frac{\partial \rho(s(\mathsf{p}), t(\mathsf{p}), \mathsf{k})}{\partial s} \\ &+ \frac{\partial t(\mathsf{p})}{\partial \psi} \sum_{\mathsf{k} \in \mathcal{Z}^2} \hat{\mathcal{T}}(\mathsf{k})\,\frac{\partial \rho(s(\mathsf{p}), t(\mathsf{p}), \mathsf{k})}{\partial t}\end{aligned} \tag{13}$$

The derivatives of the elliptical resampling filter ρ in this expression are derivative of Gaussian filters. For the filter in Figure 2 these gradient kernels are illustrated in Figure 4. The derivative is taken along the

[1] To avoid clutter, we omit the time dependence here.

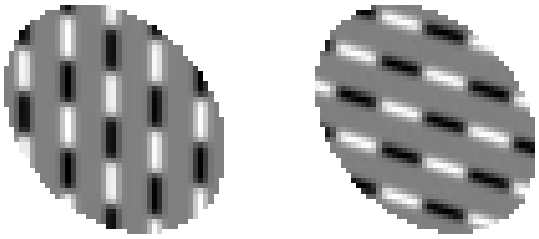

Figure 5: Gradient images

principal axes of the elliptical filter, so in general the axes of symmetry of these filters are not axis-parallel.

Convolving the texture $\hat{\mathcal{T}}$ with these gradient kernels yields two *gradient images*, illustrated in Figure 5 (corresponding to the patch in Figure 1).

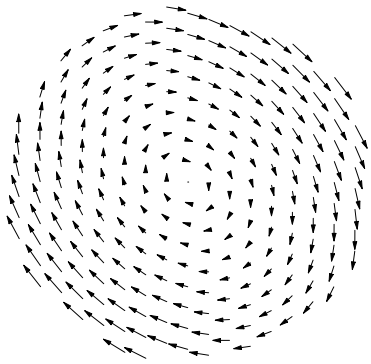

Figure 6: Vector field induced by a change in yaw ψ.

Equation (13) can now be interpreted as follows: the delta-images, representing the partial derivatives of the image with respect to one pose variable, can be obtained as a linear combination of two gradient images. The coefficients of the linear combination are *pixel dependent* and are the components of a vector field, induced by the motion of the patch. For example, the vector field induced by yaw is shown in Figure 6. Since the patch has six degrees of freedoms, there are 6 vector fields, yielding 6 different delta-images.

These vector fields correspond to the motion *in texture space* of the pre-images of the pixels, as a result of a change in one of the pose parameters. Intuitively, if one casts rays from the camera center to the surface patch, these rays intersect with the patch at specific locations. When the patch moves, these locations move as well. This is what the vector field shows.

In summary, the calculation of the measurement Jacobians H proceeds as follows: (1) calculate the gradient images; (2) calculate the vector fields; (3) combine them as in equation (13). This results in six different delta-images, or, in other words, in six elements of the Jacobian per pixel (the corresponding pixels in the delta-images). In practice, we do not calculate complete delta-images, but proceed on a pixel per pixel basis. This is useful, because -using sequential updating- randomly picked pixels can be integrated one at a time, until the motion uncertainty has dropped below a certain threshold. This yields considerable computational savings.

2.4 Updating the Texture Estimate

After the position update, the newly aligned image measurement is incorporated to refine the texture estimate $\hat{\mathcal{T}}$. The usual Kalman measurement update equations are used to update the texture, but to keep the computation tractable we make a number of approximations. Since the estimated texture is typically large (especially when working at super-resolved resolutions), it is infeasible to keep a full covariance matrix around. Instead, we neglect *all* cross-correlation terms between neighboring texture mixels, i.e. we assume a diagonal covariance matrix P_t. Furthermore, we currently neglect any cross-correlation terms between texture mixel values in $\hat{\mathcal{T}}$ and the pose $\hat{\mathsf{x}}_p$.

The measurement update equation is of exactly the same form as the equations (8-10). But now, the Jacobian H is given to us 'for free' by the texture mapping calculations. Indeed, H describes the derivative of image pixel intensity with respect to a change in texture mixel value. From equation (7) we see that this derivative is simply $\rho(\mathsf{m}^{-1}(\mathsf{p}), \mathsf{k})$.

The mixel update equations are then simple. Summarizing, we have a diagonal covariance matrix P_t, having as elements the mixel variances σ^2_{kk}, and we have the Jacobian H containing the resampling weights $W_k = \rho(\mathsf{m}^{-1}(\mathsf{p}), \mathsf{k})$. When integrating one pixel measurement at a time, the innovation $v(\mathsf{p}) = z(\mathsf{p}) - \mathcal{I}(\mathsf{p}, \hat{\mathsf{x}}_p, \hat{\mathcal{T}})$ reduces to a scalar, as does the measurement noise variance R. It can then be easily derived that the update equations (8-10) reduce to:

$$K_k = \sigma^2_{kk} W_k / (R + \sum_j \sigma^2_{jj} W_j^2) \tag{14}$$

$$\hat{\mathcal{T}}(\mathsf{k}) \Leftarrow \hat{\mathcal{T}}(\mathsf{k}) + K_k v(\mathsf{p}) \tag{15}$$

$$\sigma^2_{kk} \Leftarrow \sigma^2_{kk}(1 - W_k K_k) \tag{16}$$

Remarkably, these simple equations produce quite satisfactory super-resolved estimates of the texture $\mathcal{T}$ over time, despite the large approximation in neglecting cross-correlation terms.

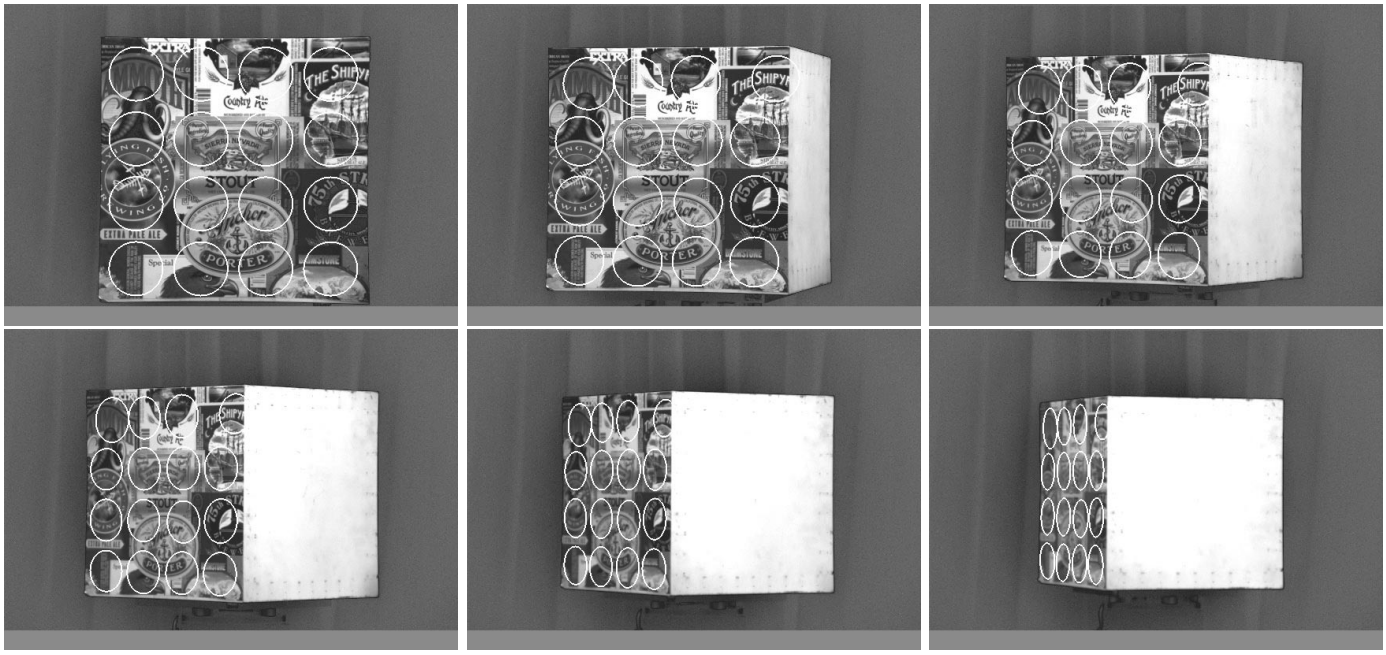

Figure 7: Frames 0,7,14,21,28, and 34 of a tracking sequence that has 16 patches tracking the texture on the cube in parallel. This sequence is available for viewing on the web at URL `http://www.cs.cmu.edu/~dellaert/research/patches.html`

3 Results and Future Work

We present some preliminary results on real image sequences, that demonstrate the viability of the approachWorking with real image sequences (as opposed to synthetic image sequences) raises a host of issues that highlight areas for future improvement.

The tracking sequence in Figure 7 shows 16 patches tracking the texture on the cube in parallel. We had precise control over the motion parameters of the cube in this 35 frame sequence. In particular, the cube moves backwards, 1 inch at a time, while simultaneously rotating by 2 degrees at each time step. Tracking 16 patches at once gives a good overview of the strengths but also of the remaining problems of the approach. It also hints at a generalization to arbitrary surfaces, as using multiple small patches one could approximate curved and other surfaces. The patches can be combined in one multiple patch model, e.g. using the oriented particle approach by Szeliski [10, 3]. As can be appreciated from the figures, all patches track the surface accurately, and in full 3D. The current implementation is not tuned for speed, but can track between 1 and 4 frames per second per surface patch.

However, the sequence brings out a number of apparent problems as well. In the beginning of the sequence, some of the patches slide over the cube surface. Only a few frames into the sequence do they lock on to the texture. We suspect that this is because the texture map is still too blurred at that point to accurately track. In addition, the sequence of images actually moves out of focus about halfway. We have no explicit model to deal with this.

Figures 8 and 9 highlight the super-resolution aspect of our approach. These images were taken from a 20 frame sequence in which the patch simply moves backwards 1 inch at a time. In Figure 8, the top left is the original image as seen by the camera (50 by 50 pixels). It is hard if not impossible to read the words in the oval. The second panel shows the initial texture state $\hat{\mathcal{T}}$ of the Kalman filter, obtained by inverse texture mapping. The third panel is the texture estimate after the first texture update step, described in Section 2.4. The bottom two rows in Figure 8 show the texture estimate over time in the remainder of the sequence. In this case, the texture map is kept at a resolution three times higher than the image. As you can see, the texture is gradually super-resolved, and halfway through the sequence we can read the words in the oval, as well as make out some other detail previously hidden. The last frame is actually a little worse than the one before that, as here too the cube moves out of focus. Finally, in Figure 9, a high-zoom exposure of the original image is compared with the texture estimate at the end of the sequence.

Figure 8: Super-resolving the texture while tracking. Top: first original image, initial estimate, and first refinement. Rows below that: texture estimates after frames 3,6,9,12,15, and 18. The texture is only refined in the central circular region, hence the blurring in the corners

4 Conclusion

We have presented a novel algorithm that can be used to simultaneously track and super-resolve the texture on a planar textured surface. We can think of many extensions and improvements to the current implementation, some of which were discussed in the previous section. Losing focus and changes in lighting are two issues our approach does not yet deal with, and provide interesting areas for future research.

However, the main issue we have not discussed is that of initialization. In our current implementation, we always assume that the initial pose of the surface is accurately known. Dealing with initialization is a hard problem, and it will be the key to making the approach applicable in real world settings.

A related issue is the generalization to arbitrary surface representations. We have already mentioned and shown how tracking multiple patches in parallel maps nicely to an oriented particle approach [10, 3]. However, other surface representations are available and might be preferable. This also means that we will be forced to relax the planar assumption, but the texture mapping measurement model we use easily lends itself to this.

Figure 9: Comparison between high zoom exposure of the original, and an enlarged version of the last panel in Figure 8.

Acknowledgements

We would like to thank Paul Heckbert for his useful comments, and Daniel Morris for introducing us to the Calibrated Imaging Lab.

This work was supported in part by USDOT under Cooperative Agreement Number DTFH61-94-X-00001 as part of the National Automated Highway System Consortium, and by the National Highway Traffic Safety Administration (NHTSA) under contract DTNH22-93-C-07023.

References

[1] P. Cheeseman, B. Kanefsky, R. Kraft, J. Stutz, and R. Hanson. Super-resolved surface reconstruction from multiple images. In G. R. Heidbreder, editor, *Maximum Entropy and Bayesian Methods*, pages 293–308. Kluwer, the Netherlands, 1996.

[2] F. Dellaert, D. Pomerleau, and C. Thorpe. Model-based car tracking integrated with a road-follower. In *Proceedings of IEEE Conference on Robotics and Automation (ICRA)*, Leuven, Belgium, 1998.

[3] P. Fua. From multiple stereo views to multiple 3d surfaces. *International Journal of Computer Vision*, 24(1):19–35, August 1997.

[4] P. S. Heckbert. Fundamentals of texture mapping and image warping. Master's thesis, University of California, Berkeley, 1989.

[5] M. La Cascia, J. Isidoro, and S. Sclaroff. Head tracking via robust registration in texture map images. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Santa Barbara, CA, June 1998.

[6] P. Maybeck. *Stochastic Models, Estimation and Control*, volume 2. Academic Press, New York, 1982.

[7] A. Patti, M. Sezan, and A. Tekalp. Superresolution video reconstruction with arbitrary sampling lattices and nonzero aperture time. *IEEE Trans. Image Processing*, 6(8):1064–1076, August 1997.

[8] D. F. Rogers. *Procedural Elements for Computer Graphics*. McGraw Hill, Boston, MA, second edition, 1998.

[9] R. R. Schultz and R. L. Stevenson. Extraction of high-resolution frames from video sequences. *IEEE Trans. Image Processing*, 5(6):996–1011, June 1996.

[10] R. Szeliski and D. Tonnesen. Surface modeling with oriented particle systems. Technical Report CRL 91/14, Digital Equipment Corporation, Cambridge Research Lab, 1991.