Finding a vine's structure by bottom-up parsing of cane edges

Tom Botterill and Richard Green
Department of Computer Science
University of Canterbury, Christchurch, NZ
Email: tom.botterill@canterbury.ac.nz

Steven Mills
Department of Computer Science
University of Otago Dunedin, NZ

***Abstract*—A vine pruning robot uses stereo cameras to build a 3D model of vines. The robot's 3D reconstruction scheme requires the 2D structure of the vine to be extracted from each image. This paper describes how the 2D structure is extracted. We propose an image grammar-based model for how a vine generates an image. We extract cane edges from each image, then apply a bottom-up parse of the cane edges to find a vine structure explaining the image. The method is efficient and accurate, and the 2D structures are complete enough that complete 3D models of vines can be reconstructed. The scheme demonstrates the power of the image grammar model for solving complex image interpretation problems.**

I. Introduction

Our team is developing a vine pruning robot that uses images from trinocular stereo cameras to make a 3D model of vines. The robot uses the 3D model to decide which canes to remove, and to plan a path for the robot arm which makes the cuts. The 3D model must be as complete and structurally-correct as possible so that the vines are pruned correctly.

The robot uses a feature-based 3D reconstruction pipeline to reconstruct a model of the vines. This paper describes the first stage of this pipeline: extraction of the structure of the vine in 2D from each image. In subsequent stages, 2D structures from multiple images are corresponded between views to give a 3D model. The 3D model which is refined in a bundle adjustment framework, and is extended as the robot moves and more vines come into view [1]. An example of three stereo frames, the extracted 2D structure, and a reconstructed 3D model are shown in Figure 1.

Each vine consists of many individual branches ('canes'), which are connected in a tree structure. Our proposed algorithm is designed to find the 2D structure of the vine present in each image, including the positions of each individual cane, and the connections between them, as shown in Figure 1. The method is robust to occlusions between canes, variations in scale, and the presence of junk such as tendrils and dead leaves. To extract the structure as accurately as possible, the system is guided by knowledge of the structure of vines, for example that canes are smoothly curved and have uniform thickness, and that the vines have an acyclic tree structure. To represent this knowledge, we use an image grammar-based model. This model is a set of rules describing how a vine

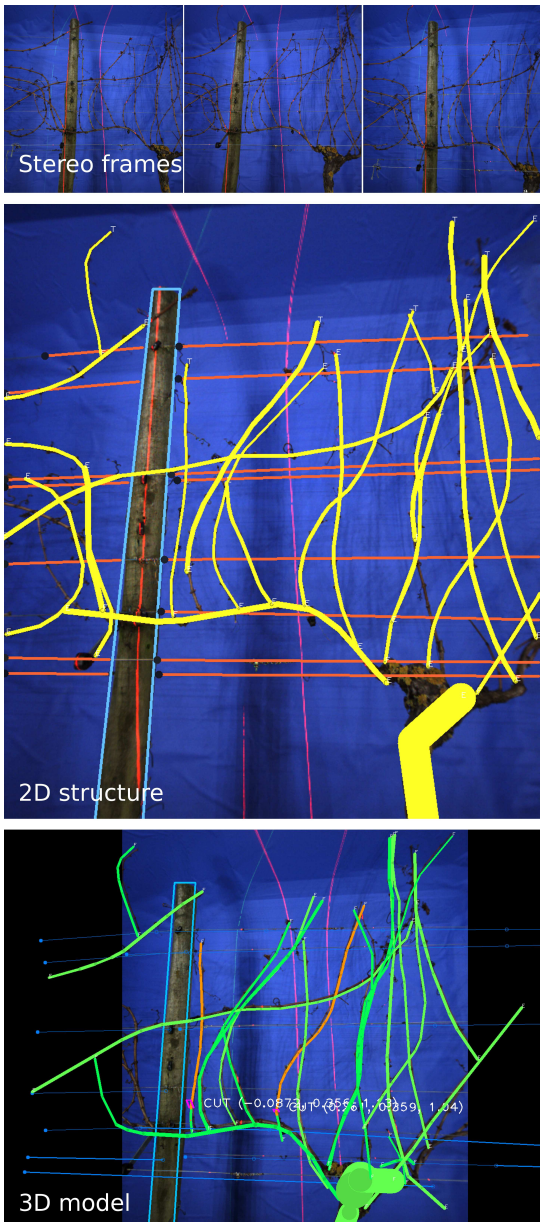

Fig. 1. The robot captures trinocular stereo frames (top), then the 2D structure of the vine is extracted from each frame (middle). The 2D structure is used to reconstruct a 3D model (bottom, shown backprojected onto one frame).

978-1-4799-0883-7/13/

generates an image. To find the vine structure explaining an image we first extract cane edges from the image, then 'parse' the image by recursively joining cane edge segments to form longer cane edges, cane segments, then complete canes. Resolving joins between these canes gives a complete 2D model of the vine. The proposed bottom-up approach is relatively simple, but is effective at extracting the 2D structure from the image: when used in the robot's 3D reconstruction system, near-complete 3D models of vines are reconstructed.

II. Background

The process of extracting a 2D network structure from an image is known as *skeletonisation*, and many different skeletonisation algorithms have been developed. Gittoes et al. [2] applied five different skeletonisation algorithms to segmented images of vines, however the resulting cane networks contained many errors including gaps in the structure, false detections due to junk in the images, artefacts from the algorithms, and detections of different structures at different scales. Bucksch et al. [3] and Huang et al. [4] recently applied state-of-the-art skeletonisation methods to images of branching plants, and in both cases many branches are incorrectly connected or missed entirely. The fundamental problem is that the models implicit in the skeletonisation algorithms are not appropriate for vines, and don't use all of the available information.

Systems for interpreting images which incorporate knowledge of objects' structure include those based on Stochastic Image Grammars [5]. These methods describe how images are generated by sets of objects, based on probabilistic rules about object co-occurrences, relationships between objects, and decompositions of objects into image primitives. The Stochastic Image Grammar model is analogous to the grammars used by computer scientists to describe languages. The aim is that images can be *parsed* in order to identify and interpret the set of objects they contain [6]. To parse an image, a set of primitives is extracted from the image (e.g. edges or segments), then a sequence of rules (e.g. explaining how combinations of parts form objects) which is likely to have generated that image is found.

Image grammar researchers aim to develop methods to represent thousands of object classes, and to automatically learn rules from vast training datasets, so that almost any image can be interpreted by a computer [5]. While these aims has not yet been realised, grammar-like models are still useful for representing knowledge in more restricted application domains. Han and Zhu [6] used a six-rule model to segment rectilinear manmade structures from cluttered scenes, and Lin et al. [7] used grammers representing a small number of object classes to detect objects despite occlusions.

More common model-based methods to locate objects in images include boosted classifiers [8] and Active Shape Models [9], however image grammars can be more powerful than these methods, as they can model objects with structural variation, they use knowledge about relationships between objects to improve accuracy, and they provide an interpretation of the scene as a whole [7]. In this paper we use a model of vines in order to interpret images of vines. The image grammar model is ideal for this purpose as vines have underlying structure that can be represented by the grammar's rules. Stochastic grammars have long been used in computer graphics for generating plant structures [10], [11]; these grammars generate acyclic tree structures with curvature, thickness variation, and fork angles sampled from approprate distributions.

III. Finding the vine structure in an image

To apply the ideas from image grammars to the problem of finding a vine's structure, we must first create a set of rules to describe how a vine generates an image. To find a vine in an image, we extract a set of primitives (cane edge segments) from the image, then apply a bottom-up parse to these to find a vine structure and parse tree which can generate the image. Each stage is detailed in the following sections.

A. Hierarchical vine model

The hierarchical vine model describing how a vine creates an image is illustrated in Figure 2. A vine consists of a set of canes connected in a tree structure. Each cane is decomposed into a sequence of shorter cane sections. Each cane section has two edges, each of which is approximated by a polyline. Each cane edge can be decomposed into a set of straight line cane edge segments. These cane edge segments are the terminal nodes in the grammar. The task we now aim to solve is to find the vine structure given a set of cane edge segments extracted from an image.

B. Extracting terminal nodes

The robot captures colour images showing the vines against a blue background (Figure 1). A variant of background subtraction which models lighting and shadows is used to segment the foreground from the background [1]. Next, wires are detected and removed from each image [1]. The edges between vines and the background are now simply the outline of the foreground segments (Figure 3).

Before extracting vine edge segments, we apply a 3×3 median filter to reduce image noise and remove very small features (tendrils), while preserving edges. We then use OpenCV's [12] contour finding function to extract the cane's outline. The Ramer-Douglas-Peucker algorithm [12], [13] is applied to simplify the edge contours into sequences of straight edge segments (polylines). Each straight edge segment forms a terminal node. We compute a thickness attribute for every cane edge segment by measuring the distance to the closest segment on the foreground side of each.

C. Parsing a cane edge image

Our proposed model describes how any vine structure generates a set of set of cane edges in an image. We now solve the inverse problem of finding a vine structure which generates this image. We find this vine structure with a bottom-up parse of the cane edge segments, as follows:

1) Recursively join cane edge segments to form cane edges (polylines)

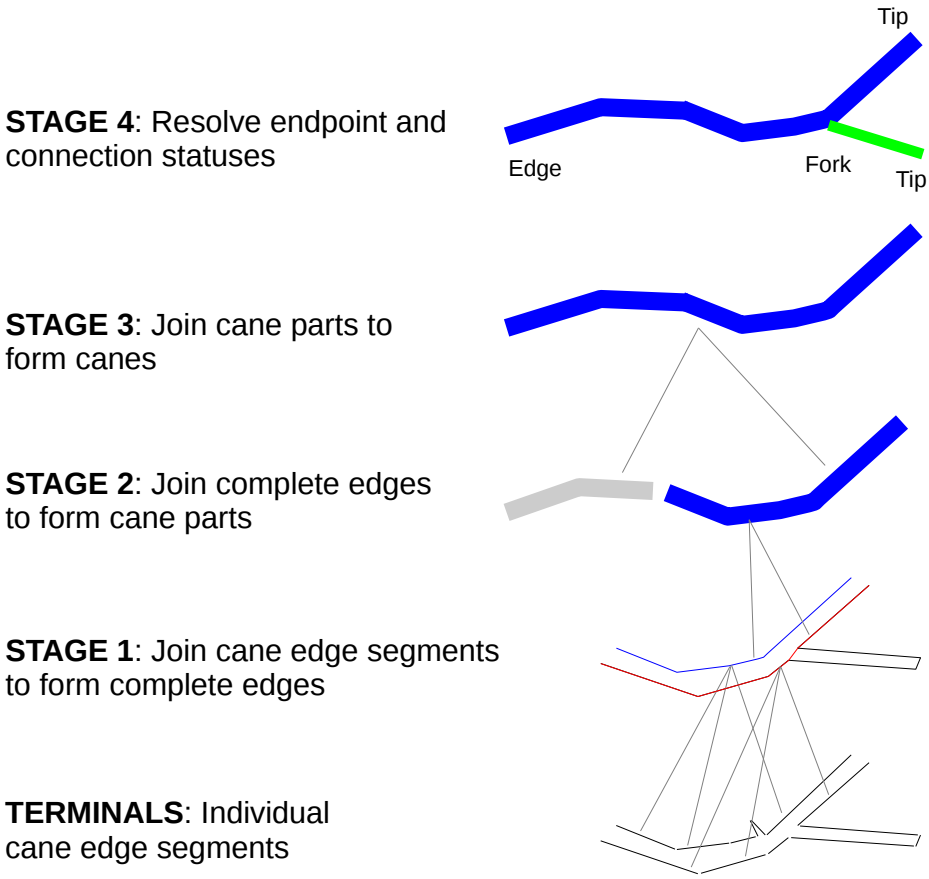

Fig. 2. Our vine model explains how a vine creates an image, and enables the image to be parsed to recover the underlying vine structure.

2) Recursively join pairs of adjacent cane edges to form cane parts
3) Recursively join cane parts to form complete canes
4) Resolve endpoint and connection statuses

The first three stages all involve repeatedly joining pairs of parts (P_1, P_2) until no more pairs can be joined. Each stage requires a binary predicate $J(P_1, P_2)$ for deciding whether two parts P_1, P_2 should be joined. These predicates are trained to make accurate join decisions for vines, i.e. they encode the knowledge about the vine's structure.

In *Stage 1*, cane edge segments (or polylines) are joined to make longer polylines. We consider pairs of cane edge segments as join candidates if they are nearby and have similar direction. We then decide if candidates should be joined by considering the following attributes:

1) The angle, offset, and separation at the join
2) The thicknesses, length, and curvature of each cane edge segment

We pose this problem as a binary classification problem: the attributes above are concatenated to make a feature vector for each pair of canes, then we use a Support Vector Machine (SVM) to classify each as 'join' or 'don't join'.

We train the SVM on hand-labelled cane edge join candidates, as shown in Figure 4. We use a ν-SVM with a Radial Basis Function kernel and set hyperparameters to maximise the two-fold cross-validation score—for details of SVMs see [14] and [15]. The SVM is very effective at correctly joining long edge sequences, as shown in Figure 5. Alternative classifiers such as random forests or neural networks could be used, however SVMs are computationally efficient and outlier tolerant, and very often these classifiers all have similar performance [16].

In *Stage 2*, we join pairs of adjacent cane edges to form cane parts (Figure 6). We use two mesurements to decide which edges to join: the median separation and maximum separation relative to the cane edge's thickness attributes. This classification task is easier than Stage 1—we could use an SVM again, but in practice a simpler model results in almost no incorrect joins being made. We assume each attribute is independent and Normal (with parameters estimated from correctly-joined edges), then compute the likelihood of each join. Pairs where the likelihood is above a threshold are joined.

In *Stage 3*, we join cane parts to form complete canes (Figure 7). These joins connect cane parts separated by junk

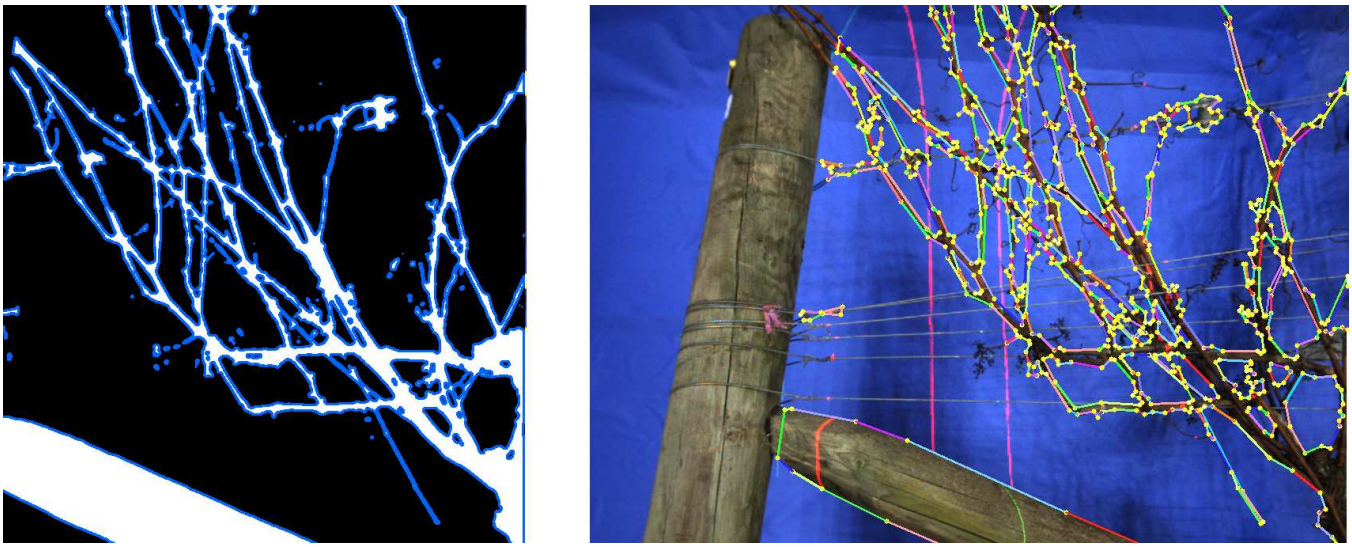

Fig. 3. Images are segmented into foreground and background, posts and wires are detected and removed, and a median filter is applied to reduce noise. The contours of the vines are extracted (left), then the Ramer-Douglas-Peucker algorithm is applied to the contours to find cane edge segment primitives (right). Individual cane edge segments are shown in different colours.

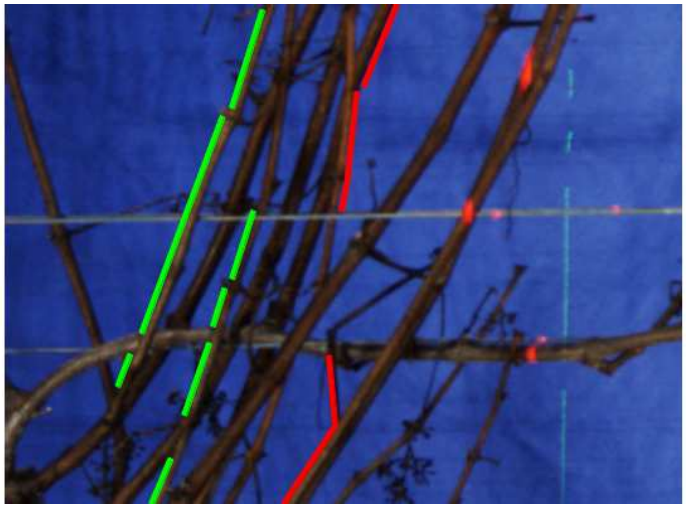

Fig. 4. The SVM decides when to join cane edges, based on a range of attributes including their thickness and curvature. This image shows typical positive training examples in green (edges which should be joined), and negative training examples in red (edges which should not be joined).

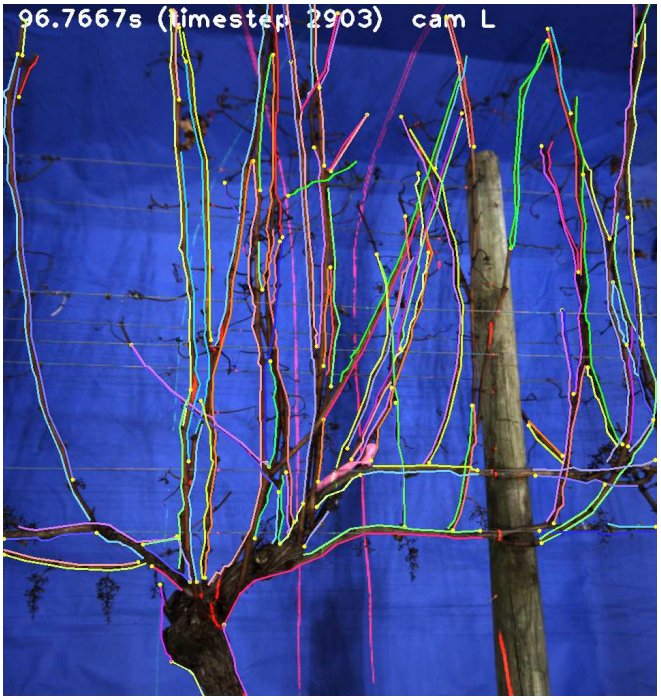

Fig. 5. Edge sequences joined by the SVM. Most cane edges are joined in long sequences corresponding to unique canes, and the SVM makes few incorrect join decisions.

in the images (dead leaves) and across complex occlusions. The attributes used to decide which cane parts to join are the cane's relative thicknesses and orientations, and the offset at the join. Again, we compute the likelihood of a join being correct, then join cane parts where the likelihood is above a threshold.

The final stage, *Stage 4*, is to label the endpoints of the extracted canes (Figure 8). We label each end 'E' for 'edge' endpoints which finish at the image edge; 'T' if the cane end is a tip finishing in free space away from other canes, and 'F' if the endpoint is a fork from another cane. Ambiguous endpoints are labelled 'E', as the 3D reconstruction framework is designed to expect some parts to be incomplete [1].

IV. Results

Our vine extraction framework is designed to provide input for building 3D models of the vines, so the best measure of its performance is how well our system can use the 2D vines to reconstruct the vines in 3D. The 3D reconstruction system corresponds 2D canes between views, and then triangulates their positions in 3D. As the robot moves, more 2D canes are

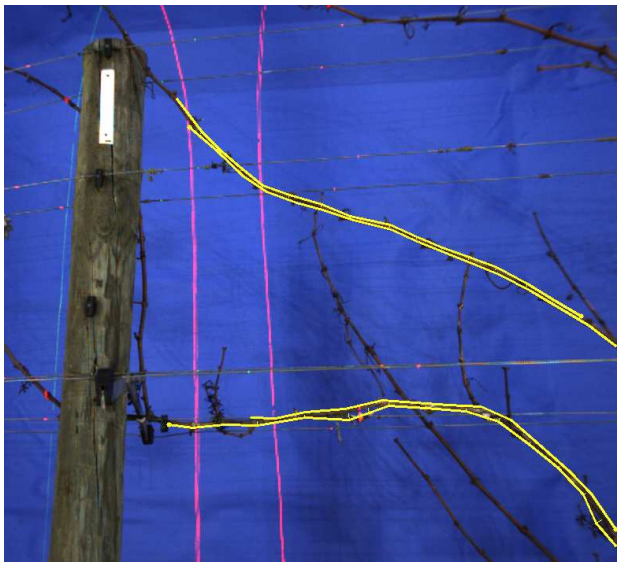

Fig. 6. Adjacent pairs of cane edges joined at *Stage 2*. These joins are comparatively easy to detect.

Fig. 7. Two pairs of cane segments which are joined at *Stage 3*.

detected. The new 2D features are either assigned to existing 3D canes, or are used to extend the 3D model. Finally, an incremental bundle adjustment jointly refines the structure and camera positions [1]. Figure 1 shows the 2D structure and 3D model of *Riesling* vines. The 3D model is almost complete—the canes that are missed are only just in view. A video of the 2D vine extraction and 3D vine reconstruction is available as supplementary material, and online at http://www.hilandtom.com/tombotterill/vines.avi.

We also reconstructed 3D models of complex *Sauvignon Blanc* vines. Extracted features, and the 3D model are shown in Figure 9. In 2D, the majority of canes are recovered correctly, but a few are missed or are incomplete. The 3D reconstruction is only 62% complete however, due to the difficulties in corresponding vines between views.

The system is implemented in C++ and runs on a single core of an Intel i7 3.6GHz processor. The entire system requires

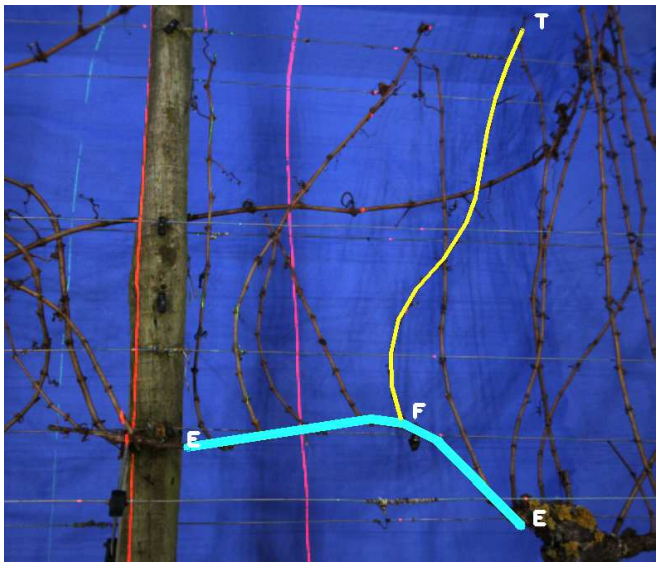

Fig. 8. Endpoint and connection statuses are resolved by comparing pairs of canes. The vertical cane forks from the bottom cane and ends at a tip. The endpoints of the bottom cane have not been resolved, so are given status 'E'.

around 3 seconds per frame. Extracting the vine structure takes between 70ms and 900ms, depending on the complexity of the vines, however the code has not yet been optimised, and optimisations will reduce these times considerably. The vine structure extraction is efficient because the purely bottom up approach rapidly reduces the number of parts which must be considered (from pixels to edge pixels to cane edges to canes).

V. Future work

The 3D reconstruction compensates for canes not being detected in every image by incrementally building the model as more canes are detected. However, the reconstruction would be more complete and more accurate if more canes were detected in each frame. Fortunately it is relatively easy to improve the performance of our grammar-based vine extractor, because at each stage of the algorithm, it is clear when decisions are correct or not. We trained the system on just two images of one vine species, and more training will improve performance substantially. Other planned improvements include:

- Investigating alternative terminal nodes for the image grammar, for example by detecting cane segments directly [17].
- Using Feature Subset Selection [18] to eliminate unnecessary descriptor components.
- Using intra-cane edges for model fitting, in addition to the cane's silhouette.
- The current bottom-up parser incorporates top-down knowledge by rejecting joins, which results in canes being missed. A more sophisticated parser could backtrack to correct earlier mistakes [6], [7].

VI. Conclusion

This paper described an image grammar-based model which describes how a vine creates an image. The model is used to

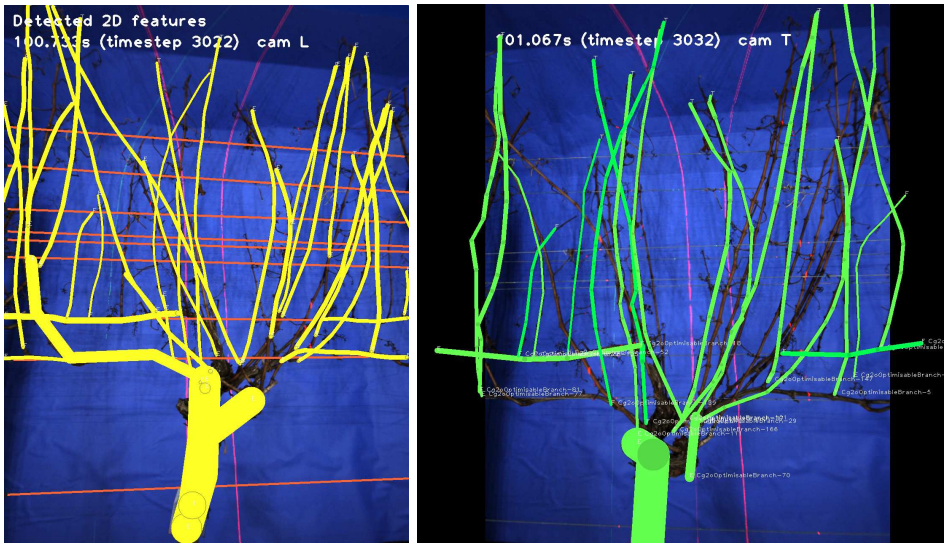

Fig. 9. 2D structure extracted from images of *Sauvignon Blanc* (left), and the corresponding 3D model (right; backprojected onto a different image).

identify the structure of vines in images by bottom-up parsing of a set of cane edges. The structure found is sufficiently accurate that the detected canes can be used in a 3D reconstruction framework. The 3D vine models are near-complete for simple vines, and are 62% complete for highly complex *Sauvignon Blanc* vines. Planned improvements to both the 2D cane extraction and the 3D reconstruction will improve the completeness further.

This paper demonstrates the effectiveness of the image grammar model for breaking down a complex image interpretation task into a sequence of decisions. Similar models could be applied in many different application domains where complex structures are extracted from images, for example in medical imaging problems.

REFERENCES

[1] T. Botterill, R. Green, and S. Mills, "Reconstructing partially visible models using stereo vision, structured light, and the g2o framework," in *Proc. Image and Vision Computing New Zealand*, 2012.

[2] W. Gittoes, T. Botterill, and R. Green, "Quantitative analysis of skeletonisation algorithms for modelling of branches," in *Proc. Image and Vision Computing New Zealand*, 2011.

[3] A. Bucksch and S. Fleck, "Automated detection of branch dimensions in woody skeletons of leafless fruit tree canopies," in *SILVILASER*, 2009.

[4] H. Huang, S. Wu, D. Cohen-Or, M. Gong, H. Zhang, G. Li, and B. Chen, "L1-medial skeleton of point cloud," *ACM Trans. Graph.*, vol. 32, no. 4, pp. 65:1–65:8, 2013.

[5] S. C. Zhu and D. Mumford, *A stochastic grammar of images*. Now Publishers Inc, 2007, vol. 2, no. 4.

[6] F. Han and S.-C. Zhu, "Bottom-up/top-down image parsing with attribute grammar," *Trans. Pattern Analysis and Machine Intelligence*, vol. 31, no. 1, pp. 59–73, 2009.

[7] L. Lin, T. Wu, J. Porway, and Z. Xu, "A stochastic graph grammar for compositional object representation and recognition," *Pattern Recognition*, vol. 42, no. 7, pp. 1297–1307, 2009.

[8] M. Jones and P. Viola, "Fast multi-view face detection," *Mitsubishi Electric Research Lab TR-20003-96*, vol. 3, p. 14, 2003.

[9] T. Cootes, C. Taylor, D. Cooper, J. Graham *et al.*, "Active shape models—their training and application," *Computer vision and image understanding*, vol. 61, no. 1, pp. 38–59, 1995.

[10] L. Quan, P. Tan, G. Zeng, L. Yuan, J. Wang, and S. B. Kang, "Image-based plant modeling," in *ACM Transactions on Graphics*, vol. 25, no. 3, 2006, pp. 599–604.

[11] P. Prusinkiewicz, A. Lindenmayer, and J. Hanan, "Development models of herbaceous plants for computer imagery purposes," in *ACM SIGGRAPH*, vol. 22, no. 4, 1988, pp. 141–150.

[12] OpenCV Computer Vision Library, n.d., http://opencv.org/.

[13] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Computer Graphics and Image Processing*, vol. 1, no. 3, pp. 244 – 256, 1972.

[14] C. Hsu, C. Chang, and C. Lin, "A practical guide to support vector classification," 2003.

[15] B. Schölkopf, A. Smola, R. Williamson, and P. Bartlett, "New support vector algorithms," *Neural computation*, vol. 12, no. 5, pp. 1207–1245, 2000.

[16] D. Meyer, F. Leisch, and K. Hornik, "The support vector machine under test," *Neurocomputing*, vol. 55, no. 1, pp. 169–186, 2003.

[17] R. D. Castaneda Marin, T. Botterill, and R. Green, "Split-and-Merge EM for vine image segmentation," in *Proc. Image and Vision Computing New Zealand*, Wellington, NZ, November 2013, pp. 1–6.

[18] A. Jain and D. Zongker, "Feature selection: Evaluation, application, and small sample performance," *Trans. Pattern Analysis and Machine Intelligence*, vol. 19, no. 2, pp. 153–158, 1997.