An image engineering approach to analysing mobile mapping data

Michael Borck
Dept. Spatial Sciences
Curtin University
m.borck@curtin.edu.au

Geoff West
Dept. Spatial Sciences
Curtin University
g.west@curtin.edu.au

Tele Tan
Dept. Mechanical Engineering
Curtin University
t.tan@curtin.edu.au

Abstract

Vehicle-based mobile mapping systems capture co-registered imagery and 3D point cloud information over hundreds of kilometres of transport corridor. Methods for extracting information from these large datasets are labour intensive. These need to be easily configured by non-expert users to process images and develop new workflows. Image Engineering provides a framework to combine known image processing, image analysis an image understanding methods into powerful applications. Such a system was built using Orange an open source toolkit for machine learning onto which image processing, visualisation and data acquisition methods were added. The system presented here enable users who are not programmers to manage image data and to customise their analyses by combining common data analysis tools to fit their needs. Case studies are provided to demonstrate the utility of the system. Co-registered imagery and depth data of urban transport corridors provided by the Earthmine dataset and laser ranging systems are used.

1 Introduction

Computer vision research is moving to the stage where quite complex systems can be used in real world applications, although in most cases the methods are turnkey. There are applications in which a non-expert user needs to configure a complex sequence of processes for their application. Such an application is the processing of co-registered imagery and 3D point cloud information acquired from a moving vehicle along transport corridors. GPS and inertial guidance allows the data to be registered to the world coordinate system enabling reasonably accurate location information to be acquired including the location of street side furniture, width of roads, power line to vegetation distances etc. Such systems can acquire enormous amounts of data quite quickly. For example the business district of Perth, Western Australia consists of 320kms of roads resulting in 120GBytes of imagery and 2GBytes of point cloud information (compressed).

In modern mobile mapping systems there are three common methods for data collection: image, point cloud, and a combination of the two through co-registered data. It is not clear which is the best as there are many algorithms for object detection that have varying measures of success. Imagery is cheap to acquire and can be interpreted by humans as well as by machine. It also contains colour information that is needed for some

In: S. Winter and C. Rizos (Eds.): Research@Locate'14, Canberra, Australia, 07-09 April 2014, published at http://ceur-ws.org

applications. Point clouds are more expensive to acquire, have no colour information and often have gaps in the data. Most point clouds for mobile mapping have low density along the direction of travel unless multiple scans are performed e.g. six, three in each direction. Finally, imagery, stereo derived point clouds and laser scanning can be combined to exploit the strength of each method.

Data from two different systems was used in this paper. The Earthmine system captures panoramic imagery and uses stereo algorithms to generate co-registered 3D point clouds. Figure 1(a) shows the Earthmine system on a car. The other system, AAM, generates depth maps from laser ranging sensor. The AAM system provides generated co-registered imagery and 3D point clouds captured from mobile scanning systems such as shown in Figure 1(b).

(a) (b)

Figure 1: (a) Earthmine system showing two sets of panorama cameras (b) MDL mobile laser scanner system with camera

Earthmine has developed a server-based system that allows the querying via location to obtain data about a particular location. The data can be processed using a number of methods e.g. randomly, from a number of known locations, or by "driving" along the capture pathway.

In many applications of computer vision, workflow is an important consideration. A user would typically read in some acquired data, process it interactively and produce the desired result. In many recognition applications, much tuning is needed requiring a user skilled in such techniques. The challenge is to produce a workflow that an non-expert user can use to configure a complex process such as object detection. To do this, a user must be able to view selected parts of the data, identify objects of interest and train a system to use the best features for recognition through a feature selection process combined with some form of pattern recognition method such as a decision tree.

This paper presents a system that gives the non-expert the opportunity to develop powerful image processing applications. The system builds on existing open source frameworks and libraries. Case studies are provided to demonstrate the utility of the system.

2 Background

Zhang (2006) defines Image Engineering (IE) as the collection of three related computer vision processes, image processing (IP), image analysis (IA), and image understanding (IU). Figure 2 show how IP, IA, and IU build up three layers of IE. Each operate on different elements. IPs primary element is the image pixel, IAs processing unit is the object, and IUs operand is a symbol or label providing meaning to an image component or segment. Each layer works with different semantic levels. From low semantic level at IP to high semantic level at IU. The three layers follow a progression of increasing abstractness and of decreasing compactness from IP to IU.

IP primarily includes the acquisition, representation, compression, enhancement, restoration, transformation and reconstruction of images. IP manipulates an image to produce another (improved) image. IA is concerned with the extraction of information from an image. IA takes an image as input and outputs data. Here, the extracted data can be the measurement results associated with specific image properties or the representative symbols of certain object attributes. IU transforms the data into descriptions allowing for decisions and actions to be taken according to the interpretation of the images.

The development of software is a complicated and tedious process, requiring highly specialized skills in systems programming (Bentrad et al., 2011). A program is usually defined as a sequence of instructions executed by a computer, so any system that executes the users actions can be considered programmable. Figure 4(a) is a fragment of python code from Demšar et al. (2004) that performs 10-fold cross validation to test a naive Bayesian classifier and k-nearest neighbors algorithm on a voting data set. Using such scripts is only suitable for experts with enough programming skills and does not allow for visual exploration and manipulation of the data.

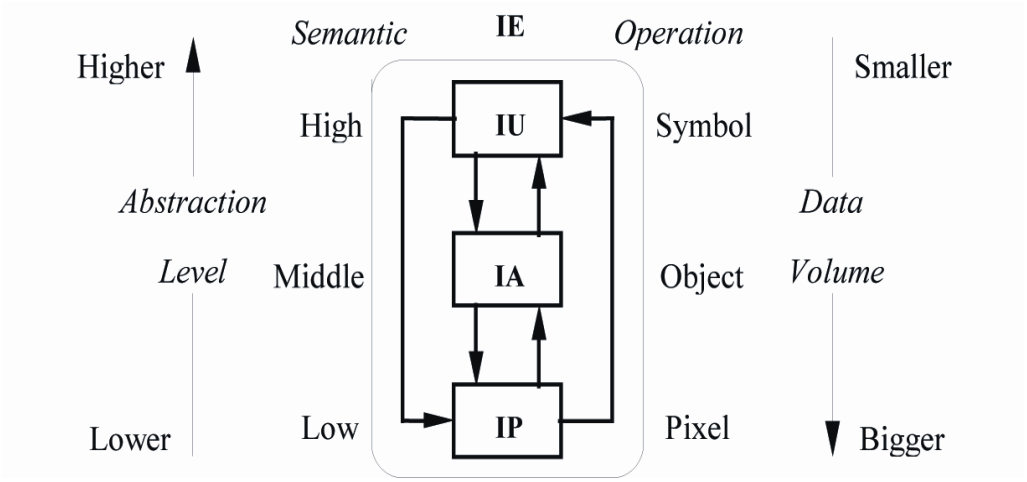


Figure 2: The three layers of image engineering (Zhang, 2006)

There are significant advantages to supplying programming capabilities in the user interfaces of a wide variety of programs (Shneiderman, 1983).

The user interface of an image processing environment is a key aspect of the proper functioning of an environment. A good user interface can significantly reduce the development effort of new image processing applications. The user interface also determines the usability of the environment to the various classes of users (Koelma and Smeulders, 1994).

One approach is the use of graphics as the programming language, or visual programming. By connecting the graphic components a user can visualise the data and the path of the data. This provides a clear expression of the flow of control in an application. The data flow metaphor is the best choice for the visual programming interface in image processing (Koelma and Smeulders, 1994). Using a visual environment provides a higher-level description of the desired actions. This makes the programming task easier even for professional programmers (Myers, 1992).

The Image Engineering system developed in this paper takes this approach. Using a visual environment allows for fast prototyping, interactive exploration of algorithms and workflow, and development of applications. The system presented exhibits characteristics and provides the benefits of visual programming, program visualisation (Myers, 1990), direct manipulation (Shneiderman, 1983), program-by-example, program-with-example (Lieberman, 2000), and demonstration interfaces (Myers, 1992).

Figure 3(a) is a fragment of python code that creates a Histogram of Gradients (HoG) (Dalal and Triggs, 2005) feature vector using the python image processing library scikit-image (van der Walt et al., 09). Figure 3(b) show the equivalent schema on the Orange canvas. Figure 3(c) opens the HoG widget showing visually options and response images. What is interesting, is the widget can process multiple images on the input stream without the user needing to understand loop constructs, unlike the code fragment that requires additional code to perform the same task. This is common for most widgets developed.

3 Underlying Technologies

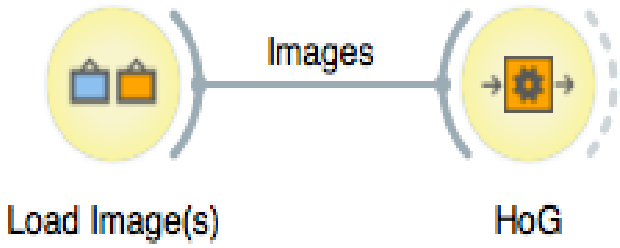
The system was built using Orange an open source toolkit for machine learning (Štajdohar and Demšar, 2013) onto which we have added image processing, visualisation and data acquisition methods recruiting functions from image processing libraries. In this paper functionality from scikit-image (van der Walt et al., 09) and OpenCV (Bradski, 2000), Scipy (Jones et al., 01) and Python (Van Rossum, 2003)have been used in the development of the system.

Orange is a general-purpose machine learning and data mining tool (Štajdohar and Demšar, 2013). It features a multi-layer architecture suitable for different kinds of users, from inexperienced data mining beginners to programmers who prefer to access the tool through its scripting interface.

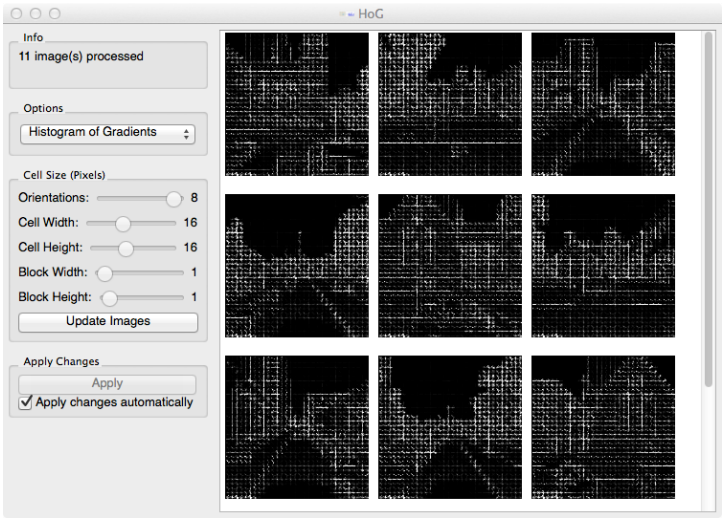
Orange Canvas provides a graphical interface for these functions. Its basic ingredients are widgets. Each widget performs a basic task, such as reading the data from a file in one of the supported formats or from a data base, showing the data in tabular form, plotting histograms and scatter plots, constructing various models and testing them, clustering the data, and so on.

```
from skimage.feature import hog
from skimage import data, color, exposure

image = color.rgb2gray(data.lena())

desc, hog_image = hog(image, orientations=8,
                      pixels_per_cell=(16, 16),
                      cells_per_block=(1, 1),
                      visualise=True)

# Process/save descriptor, perhaps display HoG image
# ....
```

(a)



(b)



(c)

Figure 3: (a) Fragment of python code calculating the HoG descriptor for an image (b) Orange schema calculating HoG descriptor for an image (c) Widget displaying HoG images a parameters of library API. Notice that the widget automatically process multiple images on the input stream.

The widgets expose the parameters of the underlying API. The advantage of widgets is in their modularity. Widgets can be connected through channels and communicate with each other by sending and receiving data. The output of one widget is used as an input for one or several other subsequent widgets. Communication channels are typed (i.e. the data type is determined to be integer, text, table, etc.) and the system establishes the proper type of data connections automatically. This property relieves the user from the need to design data structure, which is one of the greatest obstacles for lay users (Myers, 1990) (Olson et al., 1987).

A collection of widgets and their communication channels is called a schema, which is essentially a program designed by the user for a specific data analysis task. The programming process creates a schema with widgets and their connections is done visually through an easy-to-use graphic interface. Schemas can be saved and compiled into executable scripts for later reuse. The power of Orange Canvas is its interactivity. Any change in a single widget for example, loading another data set, changing the filter, modifying the logistic regression parameters can instantly propagates down the scheme.

4 Image Engineering Extension

Orange does not provide any image processing facilities but does allow you to develop your own widgets, extend scripting interface or even create your own self-contained add-ons, all seamlessly integrating with the rest of Orange, allowing components and code reuse. This allows anyone to leverage the power of the orange canvas to suit specific needs.

Figure 4(b) shows the Orange schema of the code fragment in Figure 4(a). Figure 4(c) shows the details of the *Test Learners* widget and the parameters that can be explored. Notice that unlike the code fragment details about the Area Under the Curve and Classification Accuracy and various other metrics available to the user.

The Image Engineering extension developed provides widgets designed to process image streams. The extension provides a complete workflow from image stream, to image processing, to feature vector, to machine learning, to classifier, to application. Some bespoke widgets were developed for functionality not provided by image processing libraries. For example in our systems, widgets were developed to calculate curvature, surface normals etc. from range images and statistical description of images.

Machine learning algorithms in Orange require a numerical representation of objects. If machine learning is to be performed then a widget that outputs numerical values need to be placed in the workflow as an interface between the machine learning and image processing data flow. When representing images as numerical values, the values might correspond to the pixels of an image, or the frequency of edge pixels, or perhaps a statistical description of an image region depending on the widget used. Widgets developed generally output a response image and numerical representations. Table 1 provides an overview of some of the widgets that have been developed.

5 Case Studies

Recognising objects in a scene is a primary goal of computer vision. The difficulty of this tasks depends on several factors such as: the number of objects, the complexity of the object etc. The appropriate techniques for object recognition depend on the difficulty of the task. One approach is to identify image patches, or regions of interest, that may contain objects of interest and then focus later processing, potentially computationally intensive, these regions of interest. Using a classifier, built prior, process the image patches to verify or confirm that the object of interest if present. To demonstrate the utility of the Image Engineering extension developed three case studies are provided demonstrating the above approach.

The case studies are based on the task of asset management of street furniture in urban transport corridors using the co-registered imagery and depth data. They have been simplified to focus on traffic lights but should be obvious to the reader how they could be extend to manage different assets.

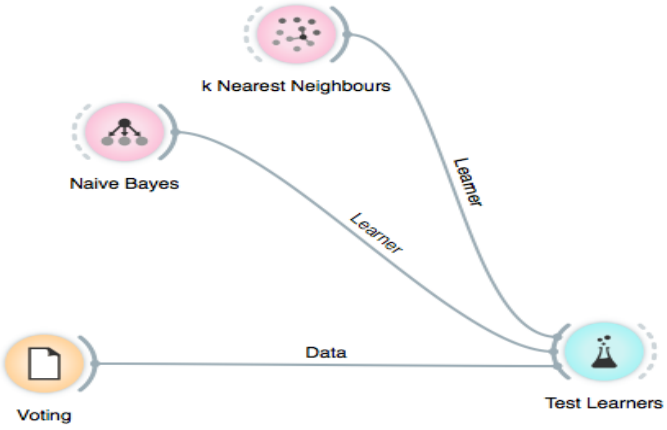
In each case study, no knowledge of how to program is required and a basic understanding of image processing is assumed. Workflows can be discovered through experimentation. Simple dragging, dropping and connecting widgets is all that is required. In one example the output from one case study is reused in another.

5.1 Locating Interesting Regions

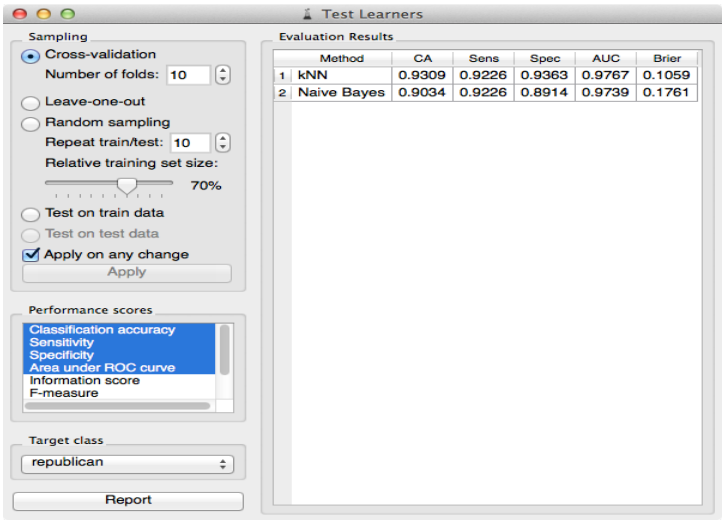
This case study uses AAM depth maps to identify interesting regions in co-registered imagery. In a scene objects exist at different depths. The intuition is that local peaks in a depth map indicate that an object is closer to the

```
from orange import BayesLearner, kNNLearner, ExampleTable
from orngTest import crossValidation
from orngStat import computrCDT

# set up the learners
bayes = BayesLearner(name='naive bayes')
knn = kNNLearner(name='knn')
learners = [bayes, knn]

# compute accuracies on data
data = ExampleTable("voting")
results = crossValidation(learners, data, folds=10)
cdt = computeCDT(results)

# output the results
# ...
```

(a)



(b)



(c)

Figure 4: (a) Fragment of python code performing 10-fold cross validation to test a nave Bayesian classifier and k-nearest neighbors algorithm on a voting data set. (B) Orange canvas schema performing 10-fold cross validation to test a nave Bayesian classifier and k-nearest neighbors algorithm on a voting data set. (c) Test Learners window exposing the parameters of the machine learning library

Table 1: A subset of widgets and associated functionality provided by image acquisition, processing and analysis extension.

Widget	Functionality
Load Image(s)	Reads image(s) from file system.
Test Image	Load one of many test images.
Earthmine	Connects to Earthmine server. Output imagery and depth maps
Channel	Select a image channel e.g. R,G,B,H,S,V. Output selected channel as an image.
Colour Space	Convert to a different colour space e.g. RGB to HSV. Output converted image.
Contours	Find the contours of an image
Depth	Find the surface normal, depth difference, curvature of the image. Output response image and histogram of response.
Edges	Compute edge map. Edge detectors available Canny (Canny, 1986), Sobel. Output edge map and edge density measure
Filter	Apply either Harr, Daubechilies or Gabor filter (Lee, 1996) to the input. Output response image and mean and variance of the filtered response.
Fourier	Convert to/from Fourier space. Output image or Fourier space image.
HoG	Calculate HoG Descriptor. Output HoG image and descriptor
Image Operations	Find Region Min/Max, Produce Histogram, Histogram Equalisation.
Interest Points	Find Harris (Harris and Stephens, 1988), SURF (Bay et al., 2008), SIFT (Lowe, 2004), or FASTER (Rosten et al., 2010) interest points. Outputs response image and the density of the interest points within the input region.
Local Binary Pattern	Calculate the LBP (Zhao and Pietikainen, 2006) of the input. Output image, mean and variance
Morphology	Apply, Dilate, Erosion, Opening, Closing, Skeleton operations on the input. Output response to morphological operation.
Salience	Calculate a saliency map using Frequency Tuned (Achanta et al., 2009), Edge based (Rosin, 2009) or Luminance and colour (Achanta et al., 2008) algorithms. Output saliency image.
Select	Select regions in an image or grid image into patches. Outputs set of regions
Segment	Segment the input using Quick Shift (Vedaldi and Soatto, 2008), SLIC (Achanta et al., 2010), and graph based (Felzenszwalb and Huttenlocher, 2004) algorithms. Output segmented image and segment labels.
Smooth	Apply either Gaussian or median filter to input image. Output smoothed image.
Statistics	Calculate and output the mean, standard deviation, skew, energy, entropy of the input image.
Threshold	Threshold image using either OTSU (Otsu, 1979), adaptive, median or user defined value. Output binary thresholded image.
Transform	Rotate, Resize or apply some other transformation. Output transformed image.

camera than its surroundings. The following method segments out things that are closer than their surroundings. A detailed description of the process can be found in Borck et al. (2014)

First a histogram equalization algorithm applied to the depth map. Then a maximal filter is used so local peaks stand out. The image is then dilated to expand the size of the local peaks. To remove the background, which essentially is unaltered in the process, the dilated image is subtracted from the output of the maximal filter. This creates an image that contains only local peaks. This local peak image is then thresholded and bounding boxes for each thresholded segment are determined. These bounding boxes are the hypothesised interesting regions and overlaid on the co-registered imagery to visually evaluate the process. For each step a widget is used to process the image stream. Figure 5 show the schema for detecting interesting regions as well as widgets for loading, dilation, background subtraction and the overlay of bounding boxes. Observe that the workflow is capable of processing multiple images without the user needing to understand and complex programming constructs such as looping or data structures.

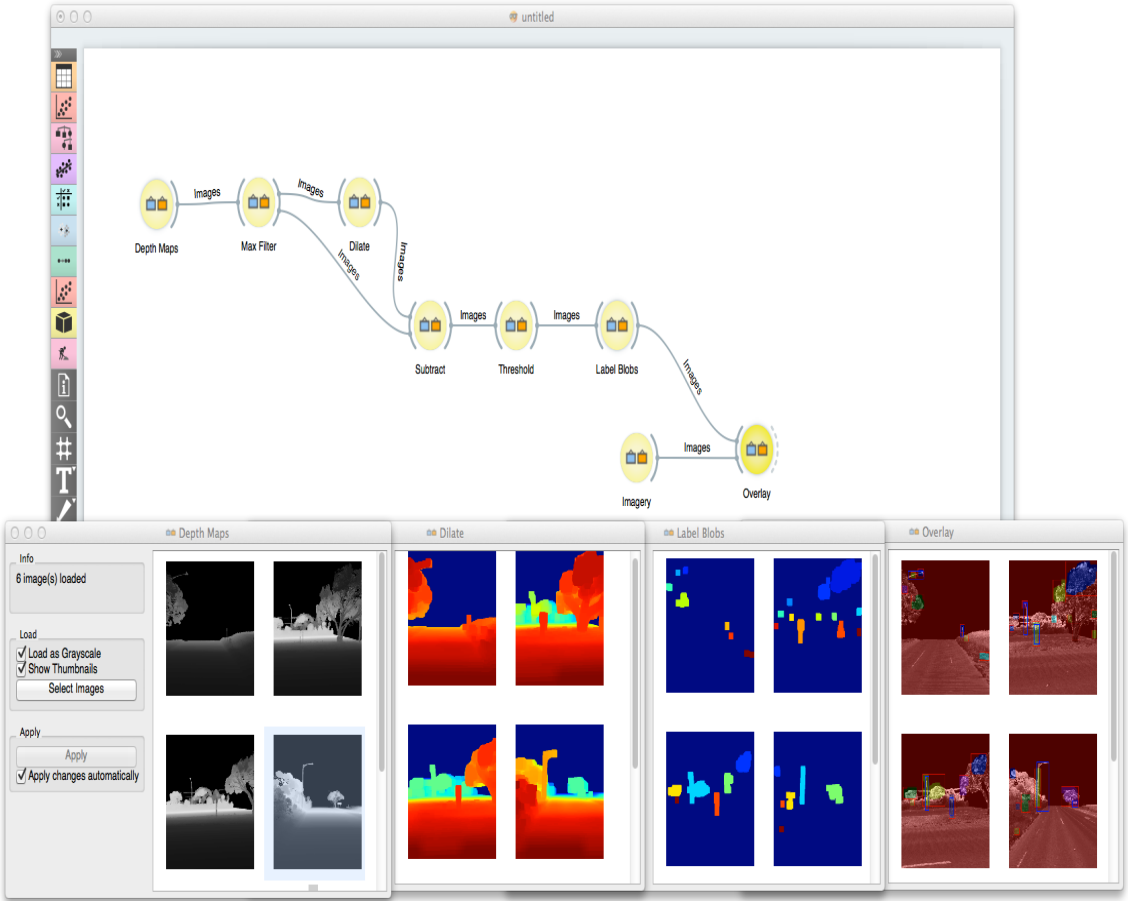


Figure 5: Schema for detecting interesting regions and widget output for loading, dilation, background subtraction and the overlay of bounding boxes.

5.2 Asset Detection

The process of using positive and negative examples with machine learning algorithms to build a object detection system is well defined in the computer vision literature. The real question is which combination of features and which machine learning algorithm is fit for the purpose. This case study describes a workflow to assess image and depth features, machine learning algorithms and infer an optimal set of features and a classifier to produce an traffic light detection system. Imagery and depth data were used from the Earthmine system. This case study implements a simplified workflow of a more complex multi class example as described in Borck et al. (2014).

The optimal classifier is determined by examining the graphs, confusion matrix, classifier metrics and other visualisations provide by the widgets use in the workflow. Once an optimal classifier has been identified it can be

saved and used in other workflows. In this case the Support Vector Machine (SVM) was the optimal classifier and is connected to the *Save Classifier* widget. Observe the two paths for the positive (Traffic Lights) and negative (Background) training examples. The feature vector is built using the HoG, Gabor and Statistics widgets. Also, since the output of the Gabor widget is an image, this need to be transformed into a descriptor. In this case a statistical summary, mean and variance, is used as the numerical representation of the Gabor feature. Notice in the schema where the output of the Gabor widget is connected to the input of the statistics widget. The HoG widget has two outputs, a descriptor and a HoG image. The HoG image allows you to visualise the feature and the descriptor is included in the feature vector. Figure 6(a) is the schema training a classifier to detect traffic lights from background. Canvas widgets can be opened by double clicking on the widget. Figure 6(b) shows open widgets for the Traffic light images, HoG feature, and the data table where each row is a feature vector. This is an example of visually exploring the workflow.

5.3 Asset Verification

This case study demonstrates a simple asset verification system and is a simplified version of a prototyped developed for a government department. Using the traffic light detection system developed earlier it is possible to label an image patch as either a traffic light or background. The Earthmine widget allows access you to specify a location. By using the Earthmine widget known locations were visited and traffic lights were extracted by manually by drawing bounding boxes around the traffic lights. A feature vector of each extraction region was built using the Hog, Gabor and Statistics widgets. The classifier developed in Section 5.2 was loaded and the newly created feature vector of unseen traffic lights were connected to the prediction widget. The prediction widget labels each feature vector as being either a traffic light or background. Figure 7 show the schema of the asset verification system.

6 Future Work

The extension developed is immediately useful for processing imagery and co-registered depth maps. Work is under way to include functionality provided by the Point Cloud Library (Rusu and Cousins, 2011). This would then provide a visual environment for the direct processing of point clouds.

There is an obvious overhead of using the visual environment, from maintaining internal data structures to updating visualisations. Many of the tasks in a schema could be executed in parallel. The functions could be distributed over several processors. The scheduling overhead would be small compared to the computational complexity of of typical image processing functions.

Widget functionality is influenced by the conceptual grouping of functions from the underlying library. This may not be the most suitable mapping onto higher level tasks performed by the widgets. Icons are used to express meaning graphically. The design of good icons is essential. Research into appropriate conceptual mapping of task and suitable expressive icons needs to be conducted. The cognitive dimensions framework of Green and Petre (1996) provides an evaluation technique that is easy to understand and quick to use and specific to visual environments and may be suitable for this purpose.

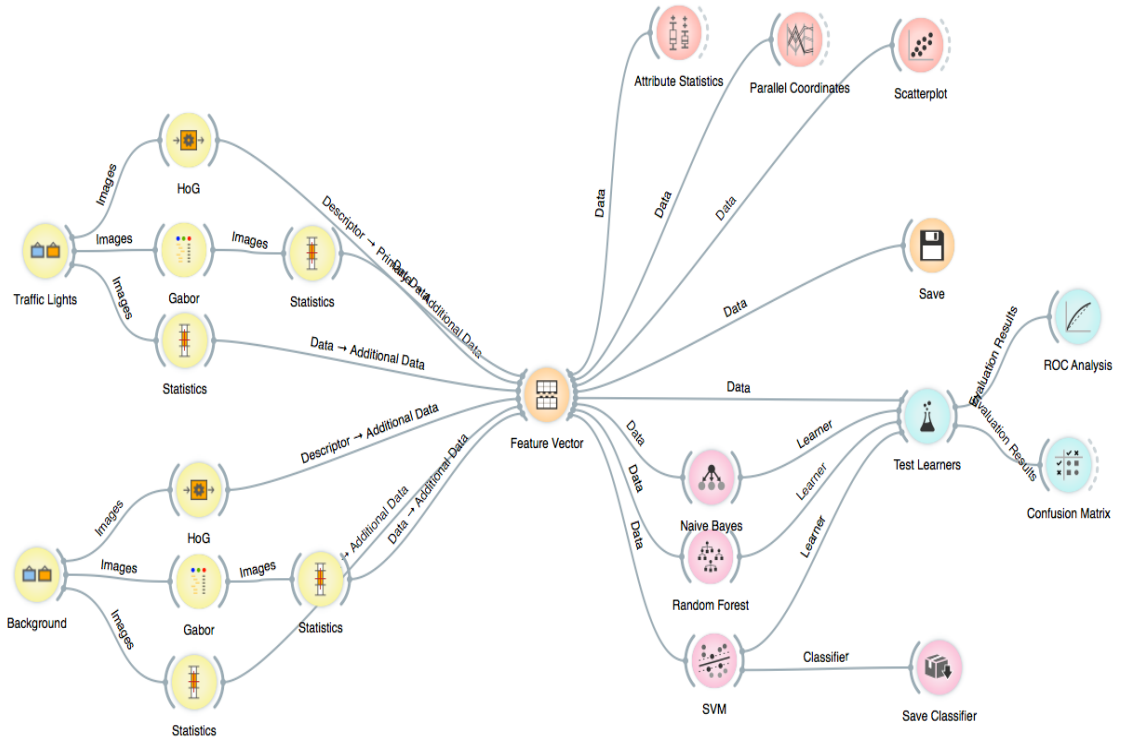
Initial feedback on prototypes developed has been positive and the perceived benefit from the user is high but a more detailed investigation into tangible benefits need to be conducted.

7 Conclusions

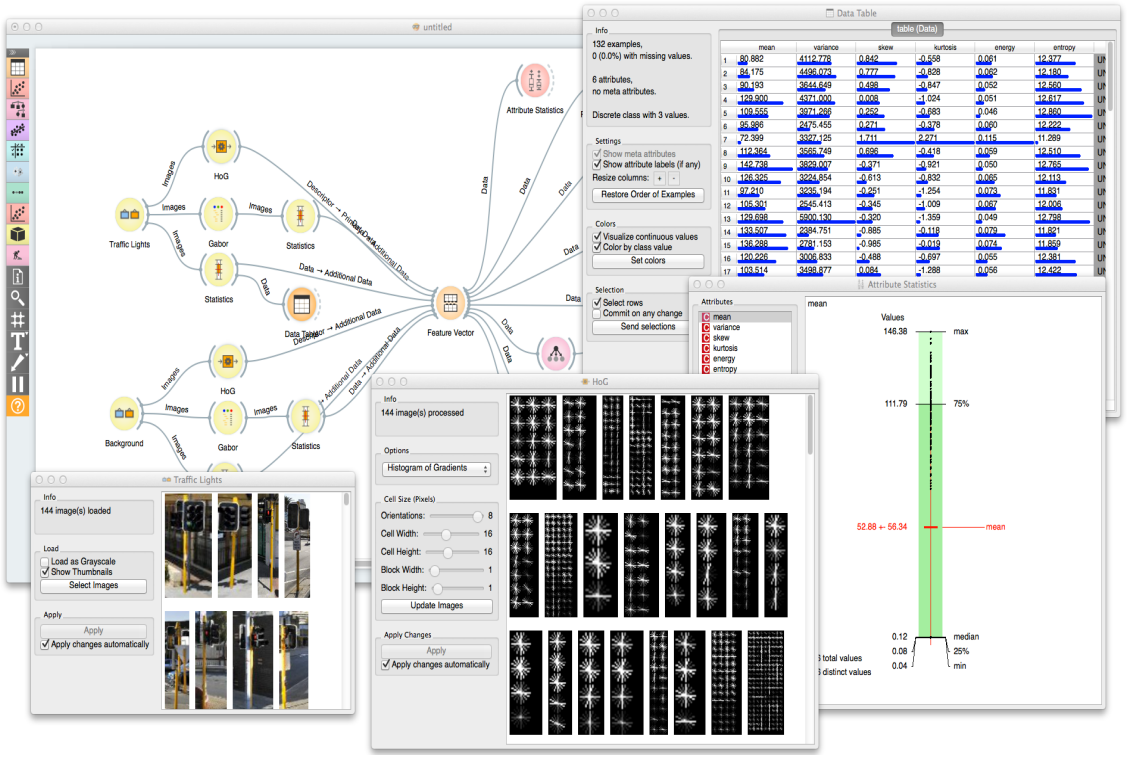
An Image Engineering extension to the Orange data analysis framework was developed. This extensions includes widgets for image acquisition, image processing and image analysis. The extension is modular and gives a non programmer user the opportunity to develop powerful image processing applications in a two-dimensional, data flow metaphor visual environment. This allows for fast prototyping and interactive exploration of algorithms and workflows without the need to learn a textual programming language. To demonstrate the utility of the system three case studies were presented. Imagery and depth data of urban transport corridors were used from the Earthmine dataset and laser ranging system.

Acknowledgment

This work is supported by the Cooperative Research Centre for Spatial Information, whose activities are funded by the Australian Commonwealths Cooperative Research Centres Programme. It provides PhD scholarship for Michael Borck and partially funds Professor Geoff West's position. The authors would like to thank John



(a)



(b)

Figure 6: (a) Example of schema training a classifier to detect traffic lights from background. (b) Example of exploring a workflow with widgets *Load Image(s)*, *HoG*, *Attribute Statistics* and *Feature Vector* opened to investigate and experiment with the workflow.

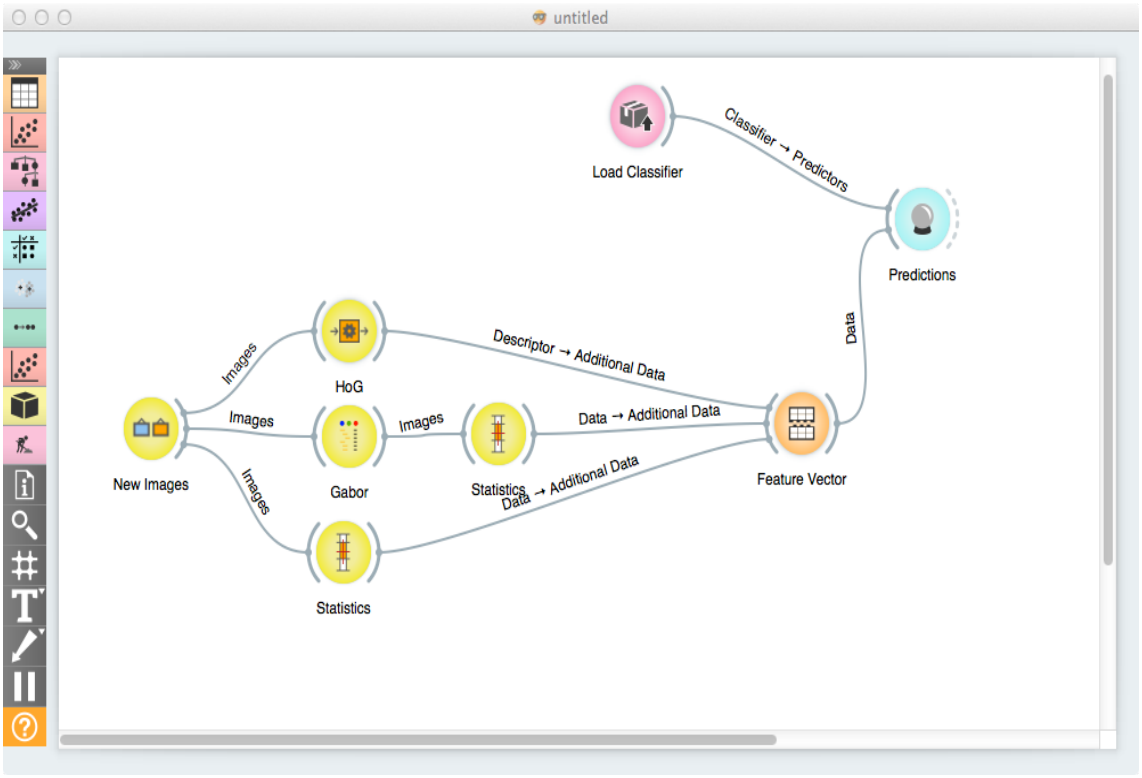


Figure 7: Schema for verifying images. First the feature vector is built then using a trained classifier predict if image regions is a traffic light.

Ristevski and Anthony Fassero from Earthmine and Landgate, WA for making available the dataset used in this work.

References

Achanta, R., F. Estrada, P. Wils, and S. Süsstrunk (2008). Salient region detection and segmentation. *Computer Vision Systems 5008*, 66–75.

Achanta, R., S. Hemami, F. Estrada, and S. Susstrunk (2009, june). Frequency-tuned salient region detection. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pp. 1597 –1604.

Achanta, R., A. Shaji, K. Smith, A. Lucchi, P. Fua, and S. Süsstrunk (2010). Slic superpixels. *École Polytechnique Fédéral de Lausssanne (EPFL), Tech. Rep 149300*.

Bay, H., A. Ess, T. Tuytelaars, and L. V. Gool (2008). Speeded-up robust features (surf). *Computer Vision and Image Understanding 110*(3), 346 – 359.

Bentrad, S., D. Meslati, et al. (2011). Visual programming and program visualization-towards an ideal visual software engineering system. *ACEEE International Journal on Information Technology 1*(3).

Borck, M., R. Palmer, G. West, and T. Tan (2014). Using depth maps to find interesting regions. *to appear in proceedings of 2014 IEE Region 10 Technical Symposium*.

Borck, M., G. West, and T. Tan (2014). Use of multiple low level features to find interesting regions. *to appear in proceedings of 3rd Internation Conference on Pattern Recognition Aplications and Methods 2014*.

Bradski, G. (2000). The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Canny, J. (1986, November). A computational approach to edge detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 8*(6), 679–698.

Dalal, N. and B. Triggs (2005, June). Histograms of oriented gradients for human detection. In C. Schmid, S. Soatto, and C. Tomasi (Eds.), *International Conference on Computer Vision & Pattern Recognition*, Volume 2, pp. 886–893.

Demšar, J., B. Zupan, G. Leban, and T. Curk (2004). Orange: From experimental machine learning to interactive data mining. In J.-F. Boulicaut, F. Esposito, F. Giannotti, and D. Pedreschi (Eds.), *Knowledge Discovery in Databases: PKDD 2004*, pp. 537–539. Springer.

Felzenszwalb, P. F. and D. P. Huttenlocher (2004). Efficient graph-based image segmentation. *International Journal of Computer Vision 59*(2), 167–181.

Green, T. R. G. and M. Petre (1996). Usability analysis of visual programming environments: a 'cognitive dimensions' framework. *Journal of Visual Languages & Computing 7*(2), 131–174.

Harris, C. and M. Stephens (1988). A combined corner and edge detector. In *Alvey vision conference*, Volume 15, pp. 50. Manchester, UK.

Jones, E., T. Oliphant, P. Peterson, et al. (2001–). SciPy: Open source scientific tools for Python.

Koelma, D. and A. Smeulders (1994). A visual programming interface for an image processing environment. *Pattern Recognition Letters 15*(11), 1099–1109.

Lee, T. S. (1996). Image representation using 2d gabor wavelets. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 18*(10), 959–971.

Lieberman, H. (2000). *Your wish is my command: Giving users the power to instruct their software.* Morgen Kaufmann.

Lowe, D. G. (2004). Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*(2), 91–110.

Myers, B. (1992). Demonstrational interfaces: A step beyond direct manipulation. *Computer 25*(8), 61–73.

Myers, B. A. (1990). Taxonomies of visual programming and program visualization. *Journal of Visual Languages & Computing 1*(1), 97–123.

Olson, G. M., S. B. Sheppard, and E. Soloway (1987). *Empirical studies of programmers: second workshop*, Volume 2. Intellect Books.

Otsu, N. (1979, January). A threshold selection method from gray-level histograms. *IEEE Transactions on Systems, Man and Cybernetics 9*(1), 62–66.

Rosin, P. L. (2009). A simple method for detecting salient regions. *Pattern Recognition 42*(11), 2363 – 2371.

Rosten, E., R. Porter, and T. Drummond (2010, Jan). Faster and better: A machine learning approach to corner detection. *Pattern Analysis and Machine Intelligence, IEEE Transactions on 32*(1), 105 –119.

Rusu, R. B. and S. Cousins (2011). 3d is here: Point cloud library (pcl). *Library*, unknown.

Shneiderman, B. (1983). Direct manipulation: A step beyond programming languages. *Computer 16*(8), 57–69.

Štajdohar, M. and J. Demšar (2013, 4). Interactive network exploration with orange. *Journal of Statistical Software 53*(6), 1–24.

van der Walt, S. et al. (2009–). scikit-image: Image processing in python.

Van Rossum, G. (2003, September). *The Python Language Reference Manual.* Network Theory Ltd.

Vedaldi, A. and S. Soatto (2008). Quick shift and kernel methods for mode seeking. In *Computer Vision–ECCV 2008*, pp. 705–718. Springer.

Zhang, Y.-J. (2006). An overview of image and video segmentation in the last 40 years. *Advances in Image and Video Segmentation*, 1–15.

Zhao, G. and M. Pietikainen (2006). Local binary pattern descriptors for dynamic texture recognition. In *Pattern Recognition, 2006. ICPR 2006. 18th International Conference on*, Volume 2, pp. 211–214. IEEE.