CF:ConeFinder Feature Extractor	
Version:	0.0.0
Date:	November 1, 2010
Purpose:	Visual tracking of orange traffic cones for application in real-time obstacle detection
Status:	documentation in progress
Author:	Steven D. Johnson

ConeFinder Feature Extractor

Contents

Technical Profile

ConeFinder is a vision function that locates orange traffice cones in a video field of view. It is used in field experimentation involving obstacle avoidance for an autonomous robotic vehicle.

This implementation uses image-processing primitives from the *OpenCV* library [5], but does not employ higher vision functionality from *OpenCV*.

Acknowledgments

The image processing algorithms used in this project were originally designed and implemented in Danko Antolovic for his *Skeyeball* tracking project [1, 2]. They were later re-implemented in Windows by Zack Rhoads and Dean Wyatt to take advantage of a PGR Firefly video camera [6]. Scott Dial developed a second, possibly different algorithm for a *CMUcam* video device with an on-board microcontroller. Ivy Want converted Dial's *ConeFinder* to use the *OpenCV*. The version documented here is based on Wang's implementation, but incorporates some aspects of Antolovic's.

This document was generated using [4].

References

[1] Danko Antolovic. Development of a real-time vision system for an autonomous model airplane. Technical Report 557, Indiana University Computer Science Department, Bloomington, Indiana, October 2002. Masters Thesis.

[2] Danko D. Antolovic, Bryce Himebaugh, and Steven D. Johnson. Skeyeball: Real-time vision system for an autonomous model airplane. In *Proceedings of the 22nd Digital Avionics Systems Conference (DASC'03)*, October 2003. Indianapolis, Indiana.

[3] Douglas Crockford. Introducing json, 2010.

[4] Steven D. Johnson. System documentation guidelines, 2009. Section 1.

[5] OpenCV User Group. Opencv welcome, July 2010.

[6] Zach Roads and Dean Wyatte. Development of a software-based real-time computer vision system for autonomous terrestrial navigation, april 2006. IUCS-CSCI-Y390 project report.

1 Requirements

SPECIFICATION 1.1 *All ERTS/vision functions satisfy these general operating requirements:*

1. Continuous operation. *The function is capable of operating periodically over a specified mission time without external intervention. Space consumption and cycle execution time are bounded as a function of image size.*

2. SyncFS mode. *For integration with* ERTS/SyncFS *[*?*,* ?*] primary output is a JSON object or string in JSON format.*

3. Dynamic image characteristics. *The function is capable of handling dynamic changes in image size and format, as necessary to perform regression testing, and input redirection.*

4. Telemetry. *The function is capable of displaying, recording, or both, a visual representation of its operation.*

1.1 ConeFinder Feature Extractor

ConeFinder is a segmentation scheme for finding contiguous regions in whose individual pixels share a common characteristic. The name "cone finder" comes from the fact that in ERTS field testing, traffic cones are used as synthetic objects in obstacle avoidance experience. Their shape and bright color make traffic cones relatively easy to identify. Figure 1 illustrate the features of interest.

SPECIFICATION 1.2 *Extracted feature elements include:*

1. Regions: *maximal contiguous regions whose pixels satisfiy a given color criterion.*

2. Boundary: *the pixels at the edge of a region.*

3. Bounding Box: *the region's smallest enclosing rectangle.*

4. Weight: *(or "mass"), a measure of region size. To be considered* significant *a region's size muse exceed a specified minimum.*

5. Centroid: *the feature's "center of mass."*

The centroid and weight values may be calculated over the entire region or just its edge. A "fuzzy" region with edge irregularities would seem heavier than one a smooth region with the same area. On the other hand, features not yet required, such as the number or configuration of corners, would be based on edge analysis.

The primary result is a *feature list*, in the form of a JSON[3] structure (Rqmt. **??**(b)) listing as many significant features as are identified.

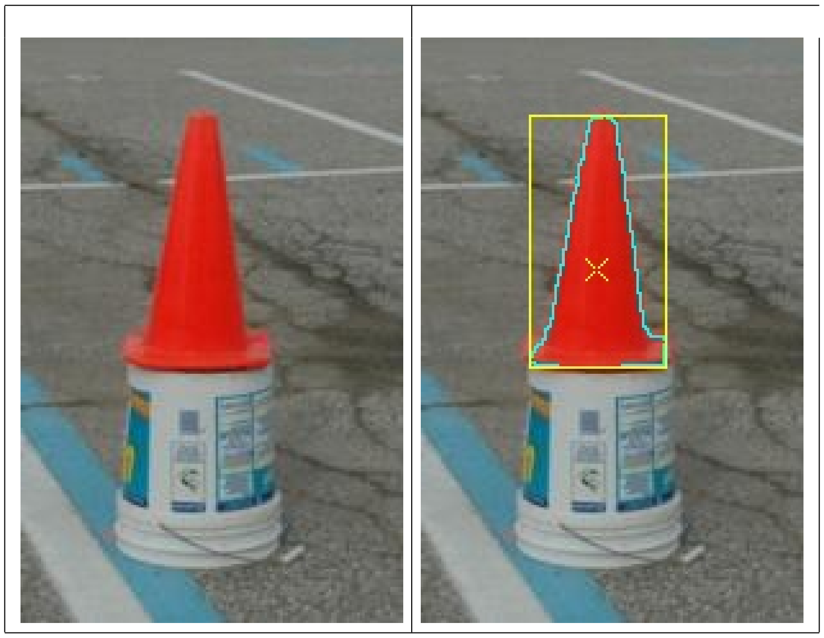


Figure 1: Traffic cone features

Specification 1.3 *For reach significant region in an image* ConeFinder *returns a* feature descriptor *listing* feature elements *specified in Requirement 1.2).*

For Figure **??**, which has just one region, the result might look like

```
["bounding_box":[388, 169, 19, 8], "center":[392, 161], "weight":114]
```

1.1.1 Image Sources

Conefinder sequentially processes a sequence of images coming either from the file system or in a live feed from a video source.

The stream of input images may be of varying size and format, within specified bounds. Image size is most likely to be 640×480 or less, and formats limited to those accepted by *OpenCV* primitives.

1.1.2 Telemetry

Like any visual feature extractor, *ConeFinder* includes options for displaying and recording its results in visual form, both interactively and in real time. Figure **??** shows an example of imagery that visualizes the features specified in Specification 1.2.

Telemetry control requires flexibility for various experimental modes, such as.

- *Sampling.* Capturing individual images under operator control, and recording. the corresponding results.
- *Recording.* Capturing images periodically at an externally controlled frame and recording the corresponding results.

Specification 1.4 *Visualization modes include*

1. *Interactive or real-time display in a window.*
2. *Saving visualized images in a directory or movie file.*

2 Design

Specification 2.1 *The implementation satisfies the general operational requirements.*

1. Continuous Operation. *Dynamically allocated storage—including structures indirectly allocated through image processing and display libraries (e.g. file I/O,* OpenCV*,* gtk*) is released within the scope in which it was obtained.*

2. SyncFS mode. *Section* **??**.

3. Dynamic image characteristics ConeFinder *relies on* OpenCV *to deal with dynamic variations in image size and format.*

4. Telemetry *Section* **??**.

2.1 Image Processing

Image processing proceeds in four stages, as depicted in Figure 2:

1. A source image is acquired.
2. Color *thresholding* is used to reduce the *source* image to monochrome regions (white in Fig. 2(b)).
3. A second pass over the monochrome image detects region edges.
4. A *connected components* algorithm is performed on the regions or their edges[1] to extract features as specified in Requirement 1.2.

Feature elements (Rqmt. 1.2) are determined as follows.

Specification 2.2

1. Fix thresholding parameters *H (hue), S (saturation), V (value, or intensity) and R (range). Let pixel P have HSV values of $\langle h, s, v\rangle$. P satisfies the* thresholding conditions *when*

$$|H - h| < R \text{ and } s \geq S \text{ and } v \geq V$$

Regions *are maximal contiguous areas in an image whose pixels all satisfy the thresholding condition.*

2. A region's edge *is a sub-region whose pixels that have at least on neighboring pixel outside the region.*

[1]Whether regions or edges are preferred, and under what conditions, remains to be determined.

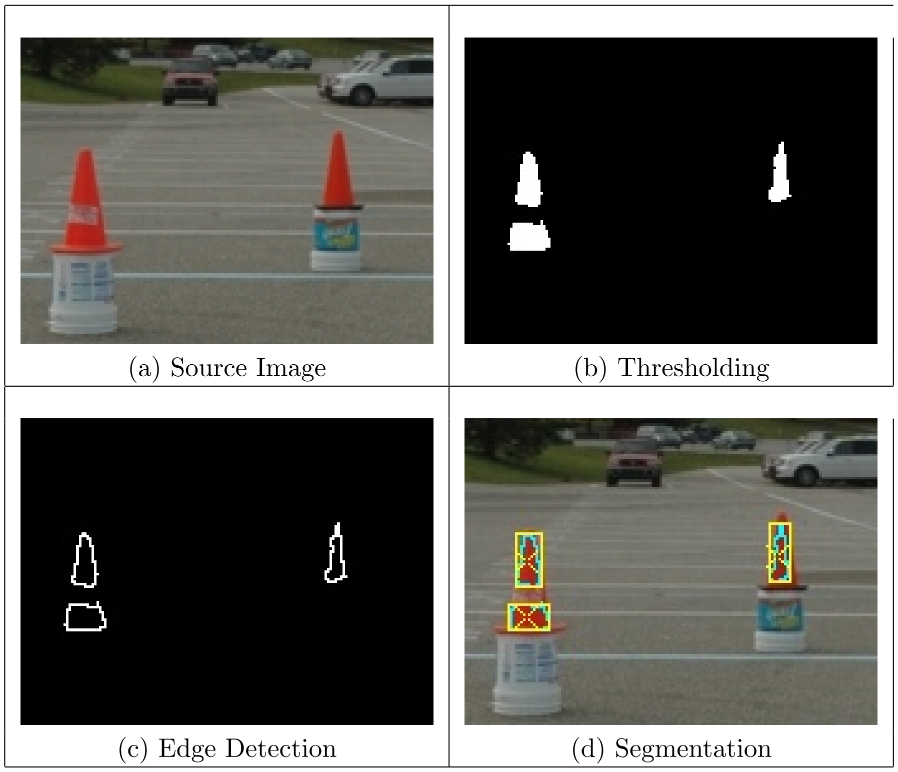

(a) Source Image (b) Thresholding (c) Edge Detection (d) Segmentation

Figure 2: Main processing stages

3. *A region's* bounding box *is determined by two diagonal corners of the smallest rectangle that fully encloses the region.*

4. *A region's* centroid *is the point determined by averaging pixels' x- and y-coordinates, or those of its edge.*

3 Implementation

Figure 3 shows the over-all organization of *ConeFinder*, whose principle entities and operation are briefly described below.

- codefinder.c contains the main program entry point.

  main() finds images to process, invokes image processing, and performs display & recording functions.

  cf_next_image() acquires source images and manages internal image structures, releasing and re-allocating when necessary.

  cf_find_cones() composes the image processing functions and performs feature-list I/O.

  cf_done() performs termination clean-up, including finally releasing dynamically allocated structures.

  src, hsv, thr, edg and dst are private references to *OpenCV* IplImage structures used by *ConeFinder*

- utilities.c

  add_tag() inserts a tag in a file name string, for instance,

  $$\texttt{aa/bb/cc.ppm} + \texttt{\_dd} \quad\Longrightarrow\quad \texttt{aa/bb/cc\_dd.ppm}$$

- ERTS/vision feature extractors:

  threshold.c

    threshold() performs thresholding on a color image, reducing it to a monochrome image.

  edge_detect.c

    edge_detect() performs edge detection on a monochrome image, eliminating the interiors of foreground regions.

  ccomponents.c

    cc_segment() finds, and returns features of, contiguous regions in a monochrome image.

    cc_reset() resets, and reallocates if necessary, local storage used by cc_segment()

    copy, E_img, S_img, B_img, C_img storage areas used in cc_segment() to perform a connected-components algorithm

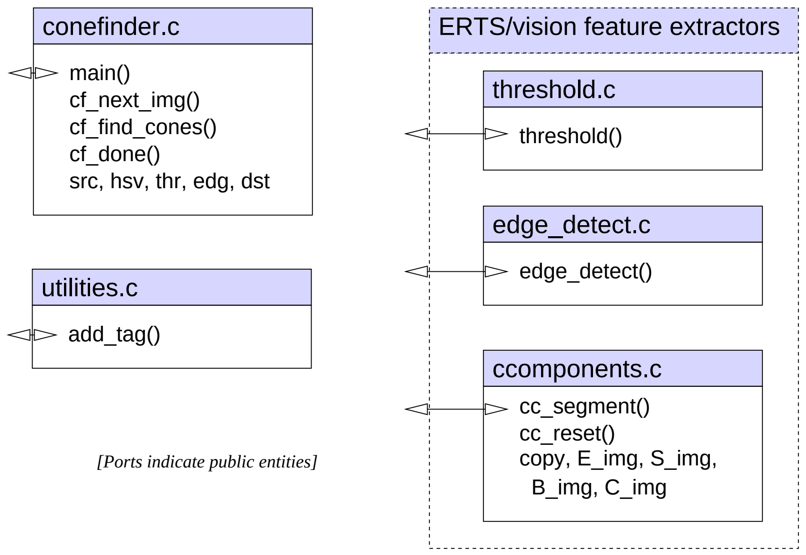


Figure 3: Organization of *ConeFinder*

3.1 Image Processing.

The conefinder.c module maintains `IplImage*` references src (source image), hsv (HSV copy of src for thresholding), thr (threshold immage), edg (edge image) and dst (destinationm or reference image). Within condfinder.c the find_cones() routine passes these references to the image processing functions, which return their imagery through them.

Each of the image processing functions has a by-reference `IplImage*` argument called REF. When ref is non-null, the procedure highlights it for use in display telemetry. That is, REF is used to show the accumulated results of image processing.

3.2 Thresholding

3.3 Edge Detection

3.4 Segmentation

3.5 Specifications

SPECIFICATION 3.1

1. Continuous operation. *See Development Test 5.1.1.*

2. SyncFS mode. *This requirement is not yet implemented.*

3. Dynamic image characteristics. *In* cf_reset() *the current source image size is saved in* `src_sz`*. Newly acquired image sizes are compared to this size, and if they differ,* `IplImage` *references are released and re-allocated. Similarly, in* cc_reset()*, dynamically allocated working storage in* `E_img`*,* `B_img`*,* `S_img` *and* `C_img` *is freed and re-allocated.*

4. Telemetry. *Examples of telemetry configurations are found in* conefinder.c

Specification 3.2

1. Regions: *Regions are determined by* threshold()*.*

2. Boundary: *Boundary edges are determined by* edge_detect()*.*

3. Bounding Box: *Bounding boxes are determined in* ccomponents()*.*

4. Weight: *Region weights are determined in* ccomponents()*. Minimum significant weight is set by the constant* `CC_MIN_WEIGHT`*.*

5. Centroid: *The centroid is determined in* ccomponents()*.*

Specification 3.3 *At present a feature list is issued to* `stdout` *from* ccomponents() *in the form of a JSON string.*

Specification 3.4

1. Interactive or real-time display in a window. *See* conefinder.c *or any* main() *routine for examples.*

2. Saving visualized images in a directory or movie file. *Not yet implemented.*

3.6 Outstanding issues

3.6.1 Errors

1. Ccomponents() sometimes returns a false feature whose bounding box surrounds chaff at the lower-left of the image.

3.6.2 To Do

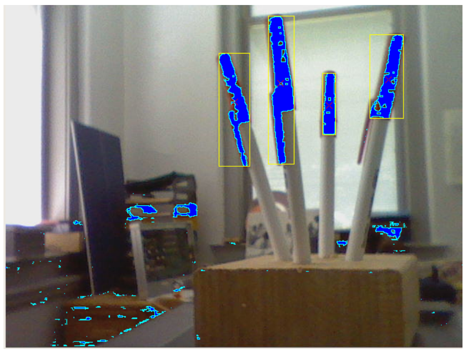
1. Change ccomponents() to return, rather than just print, a string or structure representing the feature list (Spec. 3.3.

4 Code

5 Test

5.1 Development Testing

5.1.1 Test D-1 (August 6, 2010)



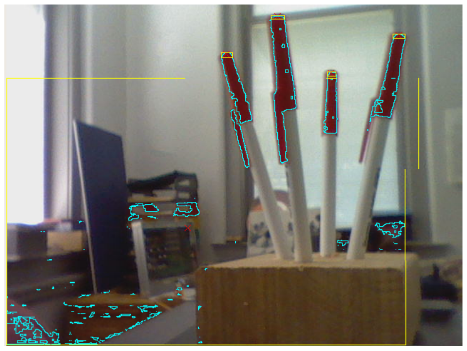
ConeFinder was run continuously for twenty-four hours on a desktop workstation. Connected-component segmentation is applied to region interiors, rather than edges. Occasionally, an erroneous bounding box appears, surrounding groups of chaff, always at the lower-left of the display (not captured in the screen shot, right).

Over the 24-hour period, thirteen warning messages were issued indicating corruption of JPEG data. The presumed causes include the web-cam interface, and timing errors at the system level.

```
Corrupt JPEG data:  premature end of data segment
Corrupt JPEG data:  16 extraneous bytes before marker 0xd9
Corrupt JPEG data:  premature end of data segment
Corrupt JPEG data:  premature end of data segment
Corrupt JPEG data:  12 extraneous bytes before marker 0xd9
Corrupt JPEG data:  13 extraneous bytes before marker 0xd9
Corrupt JPEG data:  210 extraneous bytes before marker 0xd9
Corrupt JPEG data:  83 extraneous bytes before marker 0xd9
Corrupt JPEG data:  201 extraneous bytes before marker 0xd9
Corrupt JPEG data:  premature end of data segment
Corrupt JPEG data:  premature end of data segment
Corrupt JPEG data:  15 extraneous bytes before marker 0xd9
Corrupt JPEG data:  premature end of data segment
```

5.1.2 Test D-2 (August 6, 2010)

ConeFinder Connected-component segmentation is applied to region edges, rather than interiors. Bounding boxes are wrong and an extraneous bounding box is generated, as also observed in Test D-1. These anomolies indicate problems in the connected-component implementation



5.2 Regression Testing

No regression tests are specified.

A Specification Digest

REQUIREMENTS	
1.1	All ERTS/vision functions satisfy these general operating requirements: 1. *Continuous operation.* The function is capable of operating periodically over a specified mission time without external intervention. Space consumption and cycle execution time are bounded as a function of image size. 2. *SyncFS mode.* For integration with *ERTS/SyncFS* [**?**, **?**] primary output is a JSON object or string in JSON format. 3. *Dynamic image characteristics.* The function is capable of handling dynamic changes in image size and format, as necessary to perform regression testing, and input redirection. 4. *Telemetry.* The function is capable of displaying, recording, or both, a visual representation of its operation.
1.2	Extracted feature elements include: 1. *Regions:* maximal contiguous regions whose pixels satisfiy a given color criterion. 2. *Boundary:* the pixels at the edge of a region. 3. *Bounding Box:* the region's smallest enclosing rectangle. 4. *Weight:* (or "mass"), a measure of region size. To be considered *significant* a region's size muse exceed a specified minimum. 5. *Centroid:* the feature's "center of mass."
1.3	For reach significant region in an image *ConeFinder* returns a *feature descriptor* listing *feature elements* specified in Requirement 1.2).
1.4	Visualization modes include 1. Interactive or real-time display in a window. 2. Saving visualized images in a directory or movie file.

DESIGN	
2.1	The implementation satisfies the general operational requirements. 1. *Continuous Operation.* Dynamically allocated storage—including structures indirectly allocated through image processing and display libraries (e.g. file I/O, *OpenCV*, *gtk*) is released within the scope in which it was obtained. 2. *SyncFS mode.* Section **??**. 3. *Dynamic image characteristics ConeFinder* relies on *OpenCV* to deal with dynamic variations in image size and format. 4. *Telemetry* Section **??**.
2.2	1. Fix *thresholding parameters* H (hue), S (saturation), V (value, or intensity) and R (range). Let pixel P have HSV values of $\langle h, s, v\rangle$. P satisfies the *thresholding conditions* when $$\lvert H - h\rvert < R \text{ and } s \geq S \text{ and } v \geq V$$ *Regions* are maximal contiguous areas in an image whose pixels all satisfy the thresholding condition. 2. A region's *edge* is a sub-region whose pixels that have at least on neighboring pixel outside the region. 3. A region's *bounding box* is determined by two diagonal corners of the smallest rectangle that fully encloses the region. 4. A region's *centroid* is the point determined by averaging pixels' x- and y-coordinates, or those of its edge.

Implementation	
3.1	1. *Continuous operation.* See Development Test 5.1.1. 2. *SyncFS mode.* This requirement is not yet implemented. 3. *Dynamic image characteristics.* In cf_reset() the current source image size is saved in `src_sz`. Newly acquired image sizes are compared to this size, and if they differ, `IplImage` references are released and re-allocated. Similarly, in cc_reset(), dynamically allocated working storage in `E_img`, `B_img`, `S_img` and `C_img` is freed and re-allocated. 4. *Telemetry.* Examples of telemetry configurations are found in conefinder.c
3.2	1. *Regions:* Regions are determined by threshold(). 2. *Boundary:* Boundary edges are determined by edge_detect(). 3. *Bounding Box:* Bounding boxes are determined in ccomponents(). 4. *Weight:* Region weights are determined in ccomponents(). Minimum significant weight is set by the constant `CC_MIN_WEIGHT`. 5. *Centroid:* The centroid is determined in ccomponents().
3.3	At present a feature list is issued to `stdout` from ccomponents() in the form of a JSON string.
3.4	1. *Interactive or real-time display in a window.* See conefinder.c or any main() routine for examples. 2. *Saving visualized images in a directory or movie file.* Not yet implemented.

Code

Test

B Developer Instructions

C User Instructions

Index