**Autochipping**

**Interface Control Document (ICD) V2.1**

**Created 2/15/17**

**Updated 3/11/17**

**Author: Joshua Beard**

1. **Scope of Document**

This document details the autochipping module and its usage within the HotSpotter API. This document defines the input and output typical of ordinary usage within HotSpotter, as well as command-line usage for unit testing.

1. **Concept of Utility**

Autochipping extracts a series of large rectangles, or “chips”, from a precomputed image template (a black-and-white map of an image, denoting background in black and motion in white).   
  
For each image template, the autochipping utility produces a set of chips defined by their raster position and bounds. When applied to a set of images, the utility produces these chips for each image and organizes them by image, maintaining name association.

1. **Interface**
   1. **Input**
      1. **Within HotSpotter API**
         1. *hs*

The first argument, denoted *hs*, is an instantiation of the *HotSpotterAPI* object, and is a requirement for adding each chip to the chip table.

* + - 1. *pathToTemplates*

Autochipping requires a string containing the full directory path to the location of the templates. *pathToTemplates* can use either a forwardslash (/) or backslash (\) as a directory delimiter, and will adapt based on the native operating system.

* + - 1. *exclFac*

Exclusion factor is a measure of how much of each chip is excluded in following iterations of chip extraction. The amount of each image excluded can be related to the exclusion factor by the following equation:

[](https://www.codecogs.com/eqnedit.php?latex=%5Ctext%7BPortion%20Excluded%7D%20%3D%20exclFac%5E2)

An exclusion factor of 1 ensures 100% of each chip is excluded in the next iteration, .5 ensures that a rectangle with half the width and height (resulting in a rectangle with area 25% of the original) and with the same centroid is removed.

* + - 1. *stopCrit*  
         Autochipping needs a stopping criterion, defined by *stopCrit*. The process may be approached in one of two ways: either by collecting some predetermined number of chips, or by collecting enough chips to cover some proportion of the template.   
           
         If *stopCrit* is greater than one, the autochipping module will extract that integer number of chips. The module coerces *stopCrit* to the absolute value of the nearest integer, thus ignoring improperly initialized values.  
           
         If *stopCrit* is less than one, the module will extract chips until that proportion of the template size has been captured. For instance, if *stopCrit* = 0.9, the module will continue to extract chips until 90% of the template has been captured.
      2. *skip*  
         The underlying function of autochipping is the *findLargestRects* function, which scans each line and searches for the largest rectangle contained in the template. The *skip* parameter alters this behavior, defining the number of lines that may be skipped in the search. In preliminary testing, a skip length of 8 pixels speeds up the process by about 83% and produces chips almost identical to the original.
      3. *crit*  
         The largest rectangle found by *findLargestRects* can be defined by maximizing the width, height, the area, or some combination of the three. Because the most valuable chips for recognition cover the most area, the default value for *crit* is [0, 0, 1], corresponding with width, height, and area respectively.
      4. *minSize*This is a parameter to define the minimum allowable chip size. A list of integers corresponds to pixel dimensions, and values less than one correspond with percentage of image size. The default value for the minimum size is a 1x1 square.
    1. **Standalone**

To use autochipping from the command line (for testing, debugging, etc.), use the following command from the autochip directory:

python autochip.py “complete/path/to/templates”

* 1. **Output**
     1. **Within API method**

The autochipping module returns a dictionary of lists of tuples. Each dictionary key is the name of the image, and the value corresponding with that key is the list of chips, defined by the (x, y, width, height) tuples denoting the position and bounds of the chip. An example reference is below:

>>> dictionary[“imageName”][chipNumber]

(231, 424, 678, 354)

* + 1. **Returned from API method**From outside the API method, the method returns the number of chips generated.