# FAQ-3 du manuel d’extraction des trajectoires / Trouver les objets qui bougent (ObjectFinder.jar)

Une fois que les images sont stabilisées, il faut identifier les objets mobiles

Logiciel : ObjectFinder.jar – version du 11 juillet 2013

Manuels à disposition : Object Finder Manuel

***remarque concernant les applications java***

Il est nécessaire d’avoir une machine virtuelle JAVA installée sur l’ordinateur

Si besoin la télécharger à partir du site java et choisir la configuration de l’OS (pour ordis de l’ENTPE – 64 bits : jre-8u91-windows-x64)

*Attention, il ne faut pas télécharger l’application pour les navigateurs mais celle de la MV.*

Deux possibilité pour faire tourner les applications Java

1/ exécuter le fichier .bat

Ce fichier doit contenir la ligne de commande qui lance l’application java

java -Xmx1500m -jar ObjectFinder.jar

si le nom du fichier .jar est modifié, changer son nom dans le fichier .bat

2/ utiliser l’invite de commande d’une fenêtre Ms-Dos (MicroSoft Disk Operating System)

Ouvrir la fenêtre Ms-Dos : drapeau Windows / rechercher programmes / taper cmd

Dans la fenêtre cmd.exe

i. Se placer dans la bonne partition : au départ le répertoire courant est C : \ >

pour accéder à un autre disque, taper simplement sa lettre de référence --- > F :

Nouvelle ligne F : \ >

ii. saisir le chemin d’accès au fichier .jar de ObjectFinder

rappel de quelques commandes Ms-Dos

cd .. pour redescendre dans l’arborescence ; cd nom\_dossier pour avancer ; dir pour lister le contenu du dossier ; cd\ pour revenir à la racine

penser à utiliser la touche tabulation pour ne pas saisir la totalité du nom du dossier

iii. une fois placé dans le bon répertoire, taper la commande java telle que contenue dans le fichier .bat

Pour se repérer dans les différents répertoires, il est utile d’avoir un ObjectFinder.bat à proximité des données qu’il a fabriquées, portant dans son nom le jour de la réalisation. Par exemple, chaque répertoire d’images stabilisées étant stockées dans un dossier nommé par exemple « \sCam\_1\_Zone\_2\_160911\_FM », on créera dans ce dossier le fichier .bat correspondant, que l’on nommera avec soin (en particulier la date doit être commune avec le fichier de copie d’écran de l’interface d’ObjectFinder montrant les réglages des paramètres).

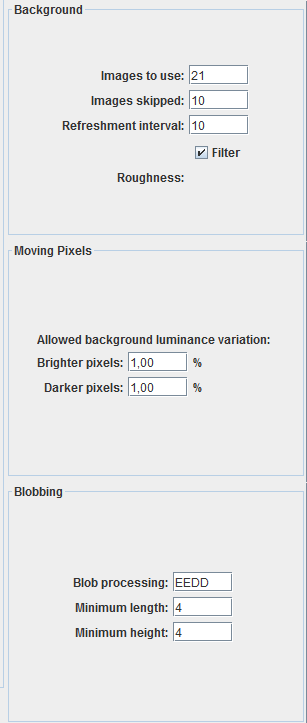
Dans le fichier .bat il faut définir la première et la dernière image à traiter. Ce peut être par exemple : FirstFile=.\images\XXX.png et LastFile=.\images\ZZZ.png si le répertoire où sont situées les images est situé dans le même répertoire que le .bat.

*Attention, pour accéder au contenu d’un fichier .bat et le lire comme un fichier txt : ouvrir (et non « exécuter en tant qu’administrateur »)*

ObjectFinder a été considérablement amélioré par Laurent Debize, lors de son séjour au LICIT dans le cadre de MOCoPo. La documentation en annexe explique la version antérieure du code. Les paramètres restent les mêmes et leurs valeurs par défaut conviennent.

Le temps de calcul est d’environ un jour pour une heure de données. Désactiver toute mise en veille, mise en veille prolongée, arrêt des disques durs (on peut laisser l’extinction du moniteur).

L’image suivante montre les valeurs par défaut des paramètres. Il faut charger un nombre d’images significativement supérieur au nombre de celles utilisées pour le calcul du background.



Contenu d’un fichier ObjectFinder.bat

Le fichier .bat est placé dans le dossier des images stabilisées que l’on va traiter

Exemple : F:\MOCoPo\Images Stabilisées\sCam\_1\_Zone\_1\_140911\_FM\_Film2

java **-Xmx1500m** -jar **ObjectFinder.jar** FirstFile=.\images\sCam\_1\_Zone\_1\_140911\_FM\_film2\_00001\_07-49-20.152.png LastFile=.\images\sCam\_1\_Zone\_1\_140911\_FM\_film2\_36239\_08-15-02.449.png

Quelques explications sur les lignes précédentes :

**-Xmx1500m :** c’est là qu’on précise l’espace mémoire à allouer à la machine virtuelle. Si cet espace n’est pas suffisant, un message d’erreur s’affiche : java heap space

**ObjectFinder.jar :** fait référence au fichier .jar situé dans le même dossier et qui permet de lancer l’application. Faire correspondre le nom du fichier .jar présent dans le dossier avec le nom inscrit dans le .bat.

Les images sont placées dans le dossier images, on spécifie alors le nom de la première et de la dernière image.

**.\images\sCam\_1\_Zone\_1\_140911\_FM\_film2\_00001\_07-49-20.152.png**

**.\images\sCam\_1\_Zone\_1\_140911\_FM\_film2\_36239\_08-15-02.449.png**

A noter que si l’application java est lancée depuis une fenêtre Ms-Dos, les lignes First et Last sont renseignées directement depuis l’application via les boutons réservés à cet effet : Select First / Select Last

# Annexes

(texte écrit par Peter Knoppers, TU Delft, NL)

## ObjectFinder Image analysis software developed at TU Delft

ObjectFinder is software that identifies moving, or changing objects in a series of images. These images must have been taken from a very stable platform or stabilized with registration software such as [ImageTracker](http://regiolab-delft.nl/?q=node/76). All images must have the same size. If the images are of a road, it helps if the road runs from left to right through the image.

Note: ObjectFinder was significantly improved, especially concerning the shadows treatment by Laurent Debize during his stay at IFSTTAR inside the MOCoPo project. The current documentation does not take those enhancements into considerations. The parameters and setup of the interface remains the same.

# Computational steps

For each image in the series:

1. Compute the background image (only performed every tenth input image)
2. Subtract the background image from the current image
3. Optimize contrast in the difference image ([equalize histogram](http://en.wikipedia.org/wiki/Histogram_equalization))
4. Find areas of in the difference images that area that belong the darkest or brightest 10% of the contrast range
5. Round the areas to remove small extensions and inner gaps
6. Remove shadows (optional)
7. Write out the results

# Details

## Computation of the background image

The background computation is performed once for each group of 10 input images. This reduces the number of times that a background images must be computed and, in case some bits of the computed background images inadvertently contain some moveable objects, these will be stationary during those 10 images. Filtering out these is easy exactly because these unwanted objects do not move.  
The background image is the median of a series of input images. for image N, this series is centered around

base = int(N / 10) \* 10 + 5

Then the series is

base + skip, base - skip, base + 2 \* skip, base - 2 \* skip, base + 3 \* skip ...

The value of skip can be changed by the user (default is 10). The maximum number of images selected can also be set by the user (default is 21).  
If this list would select images that are outside the available range, these are skipped. Up to a user-settable maximum number of images are loaded. If the *Filter* check box is checked, these images are compared to the reference image and only the 50% that are most similar to the reference image are used. Otherwise all loaded images are used. The background is computed for each pixel by taking the median value of that pixel in the used images. If the images are color images, this is done separately for the R, G and B channels.

## Detecting objects in the difference image

The difference image is computed by subtracting (pixel by pixel) the computed background image from an input image. The resulting range of pixel values is remapped to 0..255 by adding 255 and dividing by 2.  
The next step is an (optional) brightness leveling operation. For each column of pixels the median brightness is computed and corrected to 127. This appears to be beneficial for road images where the road runs from left to right and occupies most of the height of the image.  
Next an (optional) equalize histogram operation is applied. This operation computes the brightness distribution and corrects it so that each brightness value occurs in approximately the same number of pixels.  
Objects to be detected are areas that have brightness values below a minimum or above a maximum value. The default values are 10% and 90% (user changeable). For color images the filtering and object detection is done separately for each color plane.

## Blob processing

Groups of pixels in the extremes of the brightness range are detected and then cleaned up using [dilation and erosion operations](http://en.wikipedia.org/wiki/Mathematical_morphology). These operations can be carried out on the 4-neighborhood (indicated by the letter **d**, or **e**) or on the 8-neighborhood (indicated by the letter **D**, or **E**). The default set of operations is **EEDD**.  
Next, the resulting blobs that do not meet minimum size requirements are removed. By default the minimum size is 4 x 4 pixels.  
An optional *Remove shadows* operations deletes dark blobs that are adjacent to light blobs. (It is assumed that shadows are always darker than the normal background.) Separation is measured by octagonal bounding boxes. The octagonal bounding box is the smallest octagonal shape with sides at 0°, or 45° from the coordinate system axes that contains all pixels of a blob.  
Next, blobs that are very closely spaced according to their bounding boxes are merged into bigger blobs. The minimum spacing is user settable (default 4 pixels).  
For color images, these operations are done separately for each color plane and at the end, blobs in different color planes are combined when they are separated by less than the minimum separation distance.

# Use of the ObjectFinder program

The main window contains 5 areas:

1. List of files and buttons to select files and run the object finder algorithm (left)
2. Input image (right top)
3. Background image computation controls and computed background image (right, 2nd from above)
4. Threshold image computation controls and the computed threshold image (right, 3rd from abofe)
5. Blobbing controls and detected blobs image (right bottom)

# Selecting input images

Click on the *Select first image file ...* button, navigate to the directory containing the images and select the first image file.  
Then click on the *Select last image file ...* button and select the last image file.  
The list of image file name should be populated, and after a few seconds the images in the right part of the window should appear.  
If these images are larger than the available area, scroll bars appear (only) in the blobbing area. These scroll bars scroll all four images simultaneously.

# Changing the default values of the blob detection algorithm

All the default values mentioned earlier can be adjusted. The effects are computed when focus changes to another control (press the tab key, or use the mouse to select some other control). It is common that for the first few dozen images the background image contains some (partial) objects that are not part of the background. Varying the *Count* and *Skip* values may help. Usually, the results are worst for the first and last hundred images of the series because the images used to compute the background are further apart from the current image. The user may select another image in the *Input image files* list to observe how the quality of the computed background varies.

# Running the program

Click on the *Run* button to run the object finder on all images in succession and save the results. Running can be interrupted with the *Cancel* button.

*Marginal Note : the program moves from one image to the following one – about 18 images are treated in 1 minute.*

# Output file format

The results are saved as text files in the same directory as the input images files. The file names are the same, the file type is "txt". Each output file contains two lists of blobs; dark blobs and light blobs. Each group is preceded by a line "Dark blobs:" respectively "Light blobs:". Each blob is described in one line consisting of 12 tab-separated fields:

1. Left edge of blob (X coordinate), integer
2. Top edge of blob (Y coordinate), integer
3. Right edge of blob (X coordinate), integer
4. Bottom edge of blob (Y coordinate), integer
5. Top left edge of blob (X + Y coordinate), integer
6. Top right edge of blob (X - Y coordinate), integer
7. Bottom left edge of blob (X - Y coordinate), integer
8. Bottom right edge of blob (X + Y coordinate), integer
9. Center of gravity X-direction, floating point
10. Center of gravity Y-direction, floating point
11. Weight (number of active pixels within the octagonal bounding box), integer
12. Average color (RRGGBB), hexadecimal