Mask option added to the AVED project

Mask options added the 5-3-2006 on nanomia /home/aved/Jerome/
Usage: runclip -m mask.ppm my_clip.mov OR runclip -m 1-1-50-200 my_clip.mov

Project reference: AVED-05.03.2006

Here are steps and files considered to add the option to the AVED project: the Mask option allows to force the system to consider some areas as boring by flagging these areas as background after the scene segmentation.

The Mask can be defined by two way: a binary image which represent the different areas, OR parameters for a rectangle Mask given by the top-left point and its dimension.

1. Mask function creation: applyRectangleImageMask.m

```
%input_path: path of the segmented image
% mask_path: path of the mask image (should be a binary image)
%           mask pixels >= 127 -> flaged as background
%           mask pixels <  126 -> won't be touch
%
% if the segmented image size and the mask size are different,
% the smallest width and height is considered

function applyRectangleImageMask (input_path, mask_path)

   im = imread(input_path);
   im_info = imfinfo(input_path);
   mask = imread(mask_path);
   mask_info = imfinfo(mask_path);

   min_width = min(im_info.Width, mask_info.Width);
   min_height = min(im_info.Height, mask_info.Height);

   for i = 1 : min_height
      for j = 1 : min_width
         if mask(i,j) >= 127,% if this pixel has to be consider as background
            im(i,j,:) = 0;
         end
      end
   end

   [pathstr,name,ext,versn] = fileparts(input_path);

   output_path = [name '_masked' '.ppm'];
   imwrite(im,output_path,'ppm');

end
```

2. Mask function creation: applyRectangleMask.m

```
%input_path: path of the segmented image
%         x: _x position of the reference point (top, left)
%         y: _y position of the reference point (top, left)
%    height: _height of the rectangle mask
%     width: _width of the rectangle mask

function applyRectangleMask (input_path, x, y, height, width)

   im = imread(input_path);


   for i = x :x+height
      for j = y : y+width

         im(i,j,:) = 0;

      end
   end


   [pathstr,name,ext,versn] = fileparts(input_path);

   output_path = [name '_masked' '.ppm'];
   imwrite(im,output_path,'ppm');

end
```

3. Add Matlab functions to the shared library: libJeromeSegmentAlgorithms.so

The 2 matlab functions have been added to the createSharedLib script as followings:

```
mcc -W lib:libJeromeSegmentAlgorithms -g applyRectangleMask
                                         applyRectangleImageMask
                                         homomorphicCanny
                                         adaptiveThresholding
                                         backgroundCanny
```

Nb: the library has to be on /mbarivision/lib directory.

4. File mbarivision/DetectionParameters.H

```
class DetectionParameters {
public:
  //! @param itsMaskPath = path of the image which represent the mask (this one should be binair)
  std::string itsMaskPath;
  //! @param itsMaskXPosition = the x position of the reference point
  int itsMaskXPosition;
  //! @param itsMaskYPosition = the y position of the reference point
  int itsMaskYPosition;
  //! @param itsMaskWidth = mask width
  int itsMaskWidth;
  //! @param itsMaskHeight = mask height
  int itsMaskHeight;
}
```

```
class DetectionParametersModelComponent: public ModelComponent {
private:
  ModelParam<std::string> itsMaskPath;
  ModelParam<uint> itsMaskXPosition;
  ModelParam<uint> itsMaskYPosition;
  ModelParam<uint> itsMaskWidth;
  ModelParam<uint> itsMaskHeight;
}
```

5. File mbarivision/DetectionParameters.C

```
DetectionParameters::DetectionParameters()
{
  this->itsMaskPath = p.itsMaskPath;
  this->itsMaskXPosition = p.itsMaskXPosition;
  this->itsMaskYPosition = p.itsMaskYPosition;
  this->itsMaskWidth = p.itsMaskWidth;
  this->itsMaskHeight = p.itsMaskHeight;
}


DetectionParametersModelComponent::DetectionParametersModelComponent(ModelManager &mgr)
:ModelComponent(mgr, std::string("DetectionParameters"), std::string("DetectionParameters")),
        itsMaskPath(this, "MDPmaskPath", ""),
        itsMaskXPosition(this, "MDPmaskXPosition", 1),
        itsMaskYPosition(this, "MDPmaskYPosition", 1),
        itsMaskWidth(this, "MDPmaskWidth", 1),
        itsMaskHeight(this, "MDPmaskHeight", 1),
}
```
Nb: Integer variables are initialized at 1 because these variables are considered as unsigned int in others files.

```
void DetectionParametersModelComponent::exportOptions(const int whatToExport, const bool recurse)
{
   manager->requestOption(this, itsMaskPath);
   manager->requestOption(this, itsMaskXPosition);
   manager->requestOption(this, itsMaskYPosition);
   manager->requestOption(this, itsMaskWidth);
   manager->requestOption(this, itsMaskHeight);
}



void DetectionParametersModelComponent::reset(DetectionParameters *p)
{
  if(itsMaskPath.getVal().length() > 0)
    p->itsMaskPath = itsMaskPath.getVal().data();
  if(itsMaskXPosition.getVal())
    p->itsMaskXPosition = itsMaskXPosition.getVal();
  if(itsMaskYPosition.getVal())
    p->itsMaskYPosition = itsMaskYPosition.getVal();
  if(itsMaskWidth.getVal())
    p->itsMaskWidth = itsMaskWidth.getVal();
  if(itsMaskHeight.getVal())
    p->itsMaskHeight = itsMaskHeight.getVal();
}
```

6. File mbarivision/ModelOptionDefs.C

```
{ MODOPT_STRING, "MDPmaskPath", MOC_MBARI,
  "MaskPath: path to the mask image",
  "mbari-mask-path", '\0', "<file>",
  "" },
{ MODOPT_INT, "MDPmaskXPosition", MOC_MBARI,
  "MaskXPosition: x position of the mask point of reference; ",
  "mbari-mask-xposition", '\0', "<int>", "1" },
{ MODOPT_INT, "MDPmaskYPosition", MOC_MBARI,
  "MaskYPosition: y position of the mask point of reference; ",
  "mbari-mask-yposition", '\0', "<int>", "1" },
{ MODOPT_INT, "MDPmaskWidth", MOC_MBARI,
  "MaskWidth: mask width; ",
  "mbari-mask-width", '\0', "<int>", "1" },
{ MODOPT_INT, "MDPmaskHeight", MOC_MBARI,
  "MaskHeight: mask height; ",
  "mbari-mask-height", '\0', "<int>", "1" },
```

Integer variables are initialized at 1 because these variables are considered as unsigned int in others files.

7. File mbarivision/mbarivision.C

```
for (int curFrame = ifs->minframe(); curFrame <= ifs->maxframe(); ++curFrame)
  {
    /* segmentation ends here */

    if(detectionParms.itsMaskPath.length() > 0) {
      Raster::WriteRGB(bitImg,PNG,"segment");
      mxArray *inputfile = NULL;
      mxArray *mask = NULL;

      inputfile = mxCreateString("segment.png");
      mask = mxCreateString(detectionParms.itsMaskPath.c_str());

      mlfApplyRectangleImageMask(inputfile, mask);
      bitImg = Raster::ReadGray(Auto, "segment_masked.ppm");
      rv->save(bitImg, curFrame, "segment");
    }
    if (detectionParms.itsMaskXPosition != 1 || detectionParms.itsMaskYPosition != 1 ||
                                          detectionParms.itsMaskWidth != 1 || detectionParms.itsMaskHeight != 1)
      {
        Raster::WriteRGB(bitImg,PNG,"segment");
        mxArray *inputfile = NULL;
        inputfile = mxCreateString("segment.png");
        LINFO("---------------->YEPPEPEEPEP");
        mxArray *x = NULL;
        mxArray *y = NULL;
        mxArray *width = NULL;
        mxArray *height = NULL;

        x = mxCreateDoubleScalar(detectionParms.itsMaskXPosition);
        y = mxCreateDoubleScalar(detectionParms.itsMaskYPosition);
        width = mxCreateDoubleScalar(detectionParms.itsMaskWidth);
        height = mxCreateDoubleScalar(detectionParms.itsMaskHeight);

        mlfApplyRectangleMask(inputfile, x, y, width, height);
        bitImg = Raster::ReadGray(Auto, "segment_masked.ppm");
        rv->save(bitImg, curFrame, "segment");
      }
  }
```

8. File scripts/librunclip

This script is called by the runclip script.

```
# a mask can be defined by 2 way: an image OR a rectangle(1 point and its dimension)
mask=0
mask_path="" # path to the mask image
mask_x=0 # postion x of the reference point of the mask
mask_y=0 # postion y of the reference point of the mask
mask_width=0 # width of the mask
mask_height=0 # height of the mask

# Check arguments
while getopts Xfdxivj:c:t:e:p:a:m:s: option
do
  case $option in

   m)  part1tmp=${OPTARG%-*}
       part2tmp=${OPTARG#*-}
       mask=1
       if [ $part1tmp = $part2tmp ]; then # Just 1 parameter so a file is in reference
               mask_path=$part1tmp
               if [ -f $mask_path ]; then
                  echo "Process with the following mask: $mask_path"
               else
                  echo "The mask given is not valable ..."
               fi
       else # the mask is given by 1 point and its dimension
               part1=${part1tmp%-*}
               part2=${part2tmp#*-}
               mask_x=${part1%-*}
               mask_y=${part1#*-}
               mask_width=${part2%-*}
               mask_height=${part2#*-}
               echo "Process with the following mask: [$mask_x;$mask_y] width=$mask_width & height=$mask_height"
       fi;;


if [ $mask = 1 ]; then
    if [ "$mask_path" = "" ]; then # the mask is given by a point and its dimension
            useroptions="$useroptions --mbari-mask-xposition=$mask_x"
            useroptions="$useroptions --mbari-mask-yposition=$mask_y"
            useroptions="$useroptions --mbari-mask-width=$mask_widht"
            useroptions="$useroptions --mbari-mask-height=$mask_height"
    else # the mask is given by an image
            useroptions="$useroptions --mbari-mask-path=$mask_path"
    fi
fi
```