Programming Examples

Blob analysis (MIL example)

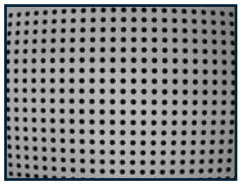
This program counts the number of objects in an image and locates the center of gravity of each objects.

"bolts.mim"

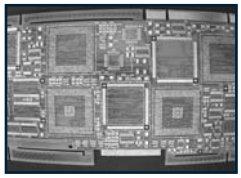
```
#include <stdio.h>
 #include <mil.h>

/* Maximum number of blobs and minimum area of blobs. */
#define MAX_BLOBS                       100L
#define MIN_BLOB_AREA                   50L
#define IMAGE_FILE                      "bolts.mim"
#define THRESHOLD_VALUE                 24L
void main(void)
{
MIL_ID   MilApplication,                /* Application identifier. */
         MilSystem,/* System identifier. */
         Millmage,                      /* Image buffer identifier. */
         BlobResult,                    /* Blob result buffer identifier. */
         FeatureList;                   /* Feature list identifier. */
long     TotalBlobs,                    /* Total number of blobs. */
         CogX[MAX_BLOBS],               /* X coordinate of center of gravity. */
         CogY[MAX_BLOBS]                /* Y coordinate of center of gravity. */
         n;                             /* Counter. */
/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, M_NULL, M_NULL, &MilImage);

/* Load source image into image buffer. */
MbufLoad(IMAGE_FILE, Millmage);

/* Binarize image. */
 MimBinarize(Millmage, MilImage, M_GREATER_OR_EQUAL, THRESHOLD_VALUE, M_NULL);

/* Allocate a blob feature list and a result buffer. */
 MblobAllocFeatureList(&FeatureList);
 MblobAllocResult(&BlobResult);

/* Enable features to be calculated. */
 MblobSelectFeature(FeatureList, M_AREA);
 MblobSelectFeature(FeatureList, M_CENTER_OF_GRAVITY_X);
 MblobSelectFeature(FeatureList, M_CENTER_OF_GRAVITY_Y);

/* Calculate selected features for each blob. */
 MblobCalculate(Millmage, M_NULL, FeatureList, BlobResult);

/* Exclude blobs whose area is too small. */
 MblobSelect(BlobResult, M_EXCLUDE, M_AREA, M_LESS_OR_EQUAL, MIN_BLOB_AREA, M_NULL);

/* Get the total number of blobs and their center of gravity. */
 MblobGetNumber(BlobResult, &TotalBlobs);
 MblobGetResult(BlobResult, M_CENTER_OF_GRAVITY_X+M_TYPE_LONG, CogX);
 MblobGetResult(BlobResult, M_CENTER_OF_GRAVITY_Y+M_TYPE_LONG, CogY);

/* Print the number of blobs and their center of gravity. */
 printf("\nThere are %ld objects in the image,\n", TotalBlobs);
 for(n=0; n< TotalBlobs; n++)
    printf("Center of gravity: X=%ld, Y=%ld.\n", CogX[n], CogY[n]);
/* Wait for a key press. */
printf("Press <Enter>.\n");
getchar();

 /* Free all allocations. */
 MblobFree(FeatureList);
 MblobFree(BlobResult);
 MappFreeDefault(MilApplication, MilSystem, M_NULL, M_NULL, Millmage);
 }
```

Calibration (MIL example)

This program illustrates camera calibration for a severe lens aberration using a calibration grid. An image acquired from the same camera is then compensated using the calibration results determined from the grid. Measurements are then performed on the calibrated image to find the width of the board in "real world" units (i.e., centimeters).



"CalGrid.mim"



"GenBoard.mim"

```
/* Regular includes. */
#include <mil.h>
#include <stdio.h>

/* Source image files specification. */
#define GRID_IMAGE_FILE                 "CalGrid.mim"
#define BOARD_IMAGE_FILE                "GenBoard.mim"

/* World description of the calibration grid. */
#define GRID_OFFSET_X                   0
#define GRID_OFFSET_Y                   0
#define GRID_OFFSET_Z                   0
#define GRID_ROW_SPACING                1
#define GRID_COLUMN_SPACING             1
#define GRID_ROW_NUMBER                 18
#define GRID_COLUMN_NUMBER              25

/* Measurement box specification */
#define MEAS_BOX_POS_X                  55
#define MEAS_BOX_POS_Y                  24
#define MEAS_BOX_WIDTH                  7
#define MEAS_BOX_HEIGHT                 425

/* Specification of the stripe constraints. */
#define APPROXIMATE_STRIPE_WIDTH    425
#define M_WIDTH_WEIGHT_FACTOR       98

/* Main application function */
void main()
{
  MIL_ID            MilApplication,           /* Application identifier. */
                      MilSystem,                  /* System identifier. */
                      MilImage,                   /* Image buffer identifier. */
                      MilCalibration,             /* Calibration identifier. */
                      MeasMarker,                 /* Measurement marker identifier. */
  double WorldWidth;

  /* Allocate defaults */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, M_NULL, M_NULL, M_NULL);

  /* Restore an image of a grid grabbed with a camera with severe lens aberration into an automatically allocated image buffer.
*/
  MbufRestore(GRID_IMAGE_FILE, MilSystem, &MilImage);

  /* Allocate a camera calibration object. */
  McalAlloc(M_DEFAULT, M_DEFAULT, &MilCalibration);

  /* Calibrate the camera with the image of the grid
   * and its world description.
   */
  McalGrid(MilCalibration, MilImage,
      GRID_OFFSET_X, GRID_OFFSET_Y, GRID_OFFSET_Z, GRID_ROW_NUMBER, GRID_COLUMN_NUMBER,
      GRID_ROW_SPACING, GRID_COLUMN_SPACING, M_DEFAULT, M_DEFAULT);
```

Calibration (continued)

```
  /* Load an image of a board grabbed with a camera with severe lens aberration and associate the calibration to the image. */
  MbufLoad(BOARD_IMAGE_FILE, MilImage);
  McalAssociate(MilCalibration, MilImage, M_DEFAULT);

  /* Allocate a measurement marker to perform measurement on the calibrated image in "real world" unit. */
  MmeasAllocMarker(MilSystem, M_STRIPE, M_DEFAULT, &MeasMarker);

  /* Set the marker measurement box. */
  MmeasSetMarker(MeasMarker, M_BOX_ORIGIN, MEAS_BOX_POS_X, MEAS_BOX_POS_Y);
  MmeasSetMarker(MeasMarker, M_BOX_SIZE, MEAS_BOX_WIDTH, MEAS_BOX_HEIGHT);

  /* Set marker orientation. */
  MmeasSetMarker(MeasMarker, M_ORIENTATION, M_HORIZONTAL, M_NULL);

  /* Set marker approximative width and the associated weight factor. */
  MmeasSetMarker(MeasMarker, M_WIDTH, APPROXIMATE_STRIPE_WIDTH, M_NULL);
  MmeasSetMarker(MeasMarker, M_WEIGHT_FACTOR+M_WIDTH, M_WIDTH_WEIGHT_FACTOR, M_NULL);

  /* Find the stripe (2 board edges) in the calibrated image and measure its width in "real world" unit.
   */
  MmeasFindMarker(M_DEFAULT, MilImage, MeasMarker, M_WIDTH);

  /* Get the world width of the marker. */
  MmeasGetResult(MeasMarker, M_WIDTH, &WorldWidth, M_NULL);

  /* Pause to show the measurement result. */
  printf("The board width is %8.4lf centimeters.\n", WorldWidth);
  printf("Press <Enter> to end.\n\n");
  getchar();

  /* Free all allocations */
  MmeasFree(MeasMarker);
  McalFree(MilCalibration);
  MbufFree(MilImage);
  MappFreeDefault(MilApplication, MilSystem, M_NULL, M_NULL, M_NULL);
}
```

Camera auto-focus (MIL example)

This program illustrates the use of the auto-focus tool to adjust camera focus by way of a motorized lens.

```
/* Regular includes. */
#include <stdio.h>
#include <mil.h>
#include <stdlib.h>

/* Lens characteristics */
#define FOCUS_MAX_NB_POSITIONS          256
#define FOCUS_MIN_POSITION              0
#define FOCUS_MAX_POSITION              255
#define FOCUS_START_POSITION            0

/* Autofocus search properties */
#define FOCUS_MAX_POSITION_VARIATION   32
#define FOCUS_MODE                      M_SMART_SCAN
#define FOCUS_SENSITIVITY               1

/* Autofocus hook function that is responsible to move the lens */
long MFTYPE MoveLensFunction(long HookType, long position, void MPTYPE *UserDataPtr);

/* Main application function                                    */
/****************************************************************/
void main(void)
{
  MIL_ID MilApplication,                /* Application identifier. */
         MilSystem,                     /* System identifier. */
         MilDisplay,                    /* Display identifier. */
         MilDigitizer,                  /* Digitizer identifier. */
         MilImage;                      /* Image buffer identifier. */
  long   FocusPos;                      /* Best focus position */

  /* Allocate defaults */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilDigitizer, &MilImage);

  /* Grab continuously. */
  MdigGrabContinuous(MilDigitizer, MilImage);

  /* Pause to show the original image. */
  printf("An autofocus operation will be performed.\n");
  printf("Press <Enter> to continue.\n");
  getchar();
  printf("Autofocusing...\n");

  /* Perform autofocus by calling the MoveLensFunction iteratively */
  MdigFocus (MilDigitizer,
             MilImage,
             M_DEFAULT,
             MoveLensFunction,
             M_NULL,
             FOCUS_MIN_POSITION,
             FOCUS_START_POSITION,
             FOCUS_MAX_POSITION,
             FOCUS_MAX_POSITION_VARIATION,
             FOCUS_MODE + FOCUS_SENSITIVITY, &FocusPos);

  /* Print the best focus position and number of iterations*/
  printf("The best focus position is %d.\n", FocusPos);
```

Camera auto-focus (continued)

```
  printf("Press <Enter> to end.\n");
  getchar();

  /* Free all allocations */
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilDigitizer, MilImage);
}

/* Autofocus hook function that is responsible to move the lens */
/*****************************************************************/
long MFTYPE MoveLensFunction(long HookType, long position, void MPTYPE *UserDataPtr)
{
  /* If focus position must be changed */
  if(HookType == M_CHANGE)
    {
    /* Move the camera lens to the specified
       position using the appropriate interface (e.g. serial port).
    */
    MoveLens(position);
    }

  return 0;
}
```

Capture and display a video sequence (MIL/MIL-Lite example)

This program grabs a sequence of images and plays it back continuously.

```
#include <stdio.h>
#include <stdlib.h>
#include <conio.h>
#include <mil.h>

#define IMAGE_WIDTH             640
#define IMAGE_HEIGHT            480
#define NBGRAB  8               /* Number of image buffers in the sequence. */

void main(void)
{
  MIL_ID MilApplication,        /* Application identifier. */
         MilSystem,             /* System identifier. */
         MilDigitizer,          /* Digitizer identifier. */
         MilDisplay,            /* Display identifier. */
         MilImage[NBGRAB],      /* Number of image buffers in the sequence. */
         MilImageDisp;          /* Display image buffer identifier. */

  long n;

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilDigitizer, M_NULL);

  /* Allocate sequence storage image buffers. */
  for (n=0; n<NBGRAB; n++)
  {
  MbufAlloc2d(MilSystem, IMAGE_WIDTH, IMAGE_HEIGHT, 8L+M_UNSIGNED, M_IMAGE+M_GRAB, &MilImage[n]);
  }

  /* Allocate a display image buffer. */
  MbufAlloc2d(MilSystem, IMAGE_WIDTH, IMAGE_HEIGHT, 8L+M_UNSIGNED,
              M_IMAGE+M_GRAB+M_PROC+M_DISP, &MilImageDisp);

  /* Display activation. */
  MbufClear(MilImageDisp, 0x0);
  MdispSelect(MilDisplay, MilImageDisp);

  /* Print a message. */
  printf("Press enter to record the sequence.\n");
  getchar();

  /* Grab the sequence. */
  for (n=0; n<NBGRAB; n++)
    {
    MdigGrab(MilDigitizer, MilImage[n]);
    }

 /* Play the sequence until a key is pressed. */
 while( !kbhit() )
    {
     /* Play the sequence once. */
     for (n=0; n<NBGRAB; n++)
        {
         /* Copy one image to the screen. */
         MbufCopy(MilImage[n],MilImageDisp);
        }
    }

 /* Free image buffers. */
 MbufFree(MilImageDisp);
 for (n=0; n<NBGRAB; n++)
    {
    MbufFree(MilImage[n]);
    }

/* Free defaults. */
MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilDigitizer, M_NULL);
}
```

Note: Playback can be synchronized with the display.

Code Reader (MIL example)

This program decodes a DataMatrix code. The string is read and then printed to the screen.

"excode.mim"

```
#include <stdio.h>
#include <string.h>
#include <mil.h>

/* Target image character specifications. */
#define CHAR_IMAGE_FILE          "excode.mim"

/* Maximum length of the string to read or draw (null terminated) */
#define STRING_LENGTH                    64L

/* Threshold value */
#define THRESHOLD_VALUE                  128L

void main(void)
{
  MIL_ID MilApplication,                 /* Application identifier. */
         MilSystem,                      /* System identifier. */
         MilDisplay,                     /* Display identifier. */
         MilImage,                       /* Image buffer identifier. */
         MatrixCode;                     /* DataMatrix code identifier */

  char   ResultString[STRING_LENGTH];    /* Array of characters read. */

  /* Allocate defaults */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, M_NULL);

  /* Restore source image into an automatically allocated image buffer. */
  MbufRestore(CHAR_IMAGE_FILE, MilSystem, &MilImage);

  /* Display the image buffer. */
  MdispSelect(MilDisplay, MilImage);

  /* Allocate CODE object */
  McodeAlloc(MilSystem, M_DATAMATRIX, M_DEFAULT, &MatrixCode);

  /* Set threshold value */
  McodeControl(MatrixCode, M_THRESHOLD, THRESHOLD_VALUE);

  /* Pause to show the original image. */
  printf("This program will decode a DataMatrix code.\n");
  printf("Press <Enter> to continue.\n");
  getchar();

  /* Read code from image */
  McodeRead(MatrixCode, MilImage, M_DEFAULT);

  /* Get decoded string */
  McodeGetResult(MatrixCode, M_STRING, ResultString);

  printf("The decoded string is : %s\n", ResultString);
  printf("Press <Enter> to end.\n");
  getchar();

  /* Free all allocations. */
  McodeFree(MatrixCode);
  MbufFree(MilImage);
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, M_NULL);
}
```

Digitizer allocation and control (MIL/MIL-Lite example)
This program illustrates continuous image capture to display.

```
#include <stdio.h>
#include <mil.h>

void main(void)
{
  MIL_ID MilApplication,        /* Application identifier. */
         MilSystem,             /* System identifier. */
         MilDisplay,            /* Display identifier. */
         MilCamera,             /* Camera identifier. */
         MilImage;              /* Image buffer identifier. */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilCamera, &MilImage);

  /* Grab continuously. */
  MdigGrabContinuous(MilCamera, MilImage);

  /* When a key is pressed, halt. */
  printf("Continuous grab in progress. Adjust your camera and\n");
  printf("press <Enter> to stop grabbing.");
  getchar();

  /* Stop continuous grab. */
  MdigHalt(MilCamera);
  printf("\nDisplaying the last grabbed image.\n");
  printf("Press <Enter> to end.");
  getchar(

  /* Release defaults. */
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilCamera, MilImage);
}
```

Displaying a MIL buffer under Windows (MIL/MIL-Lite example)

This program could form the core of a window-based MIL application. It displays a MIL image buffer in a user specified window. The MIL buffer is initialized with text drawn using MIL graphic functions.

Note: For simplicity, the program entry point is not included.

```
void MilWindowsApplication(HWND UserWindowHandle)
{
  MIL_ID  MilApplication,          /* MIL Application identifier. */
          MilSystem,               /* MIL System identifier. */
          MilDisplay,              /* MIL Display identifier. */
          MilImage;                /* MIL Image buffer identifier. */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, &MilImage);

  /* Select the image buffer to be displayed into the specified user window. */
  MdispSelectWindow(MilDisplay, MilImage, UserWindowHandle);
```

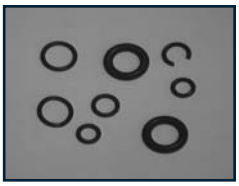
Digitizer allocation and control (MIL/MIL-Lite example)

This program illustrates continuous image capture to display.

```
#include <stdio.h>
#include <mil.h>

void main(void)
{
  MIL_ID MilApplication,           /* Application identifier. */
         MilSystem,                /* System identifier. */
         MilDisplay,               /* Display identifier. */
         MilCamera,                /* Camera identifier. */
         MilImage;                 /* Image buffer identifier. */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, &MilCamera, &MilImage);

  /* Grab continuously. */
  MdigGrabContinuous(MilCamera, MilImage);

  /* When a key is pressed, halt. */
  printf("Continuous grab in progress. Adjust your camera and\n");
  printf("press <Enter> to stop grabbing.");
  getchar();

  /* Stop continuous grab. */
  MdigHalt(MilCamera);
  printf("\nDisplaying the last grabbed image.\n");
  printf("Press <Enter> to end.");
  getchar(

  /* Release defaults. */
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilCamera, MilImage);
}


  /* Print a string in the image buffer using MIL. */
  MgraText(M_DEFAULT, MilImage, 176L, 210L, " ----------- ");
  MgraText(M_DEFAULT, MilImage, 176L, 235L, " Welcome to MIL !!! ");
  MgraText(M_DEFAULT, MilImage, 176L, 260L, " ----------- ");
```

Edge Finder (MIL example)

This program illustrates edge extraction with exclusion. Edges are located in the target image, some results are excluded based predefined criteria (edge features), final results (drawn edges and total count) are printed to screen.



"Seals.mim"

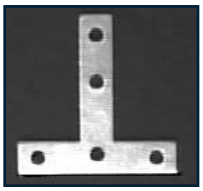
```
/* Regular includes. */
#include <mil.h>
#include <stdio.h>
#include <conio.h>

/* Source MIL image file specifications. */
#define CONTOUR_IMAGE              M_IMAGE_PATH MIL_TEXT("Seals.mim")
#define CONTOUR_MAX_RESULTS        100L
#define CONTOUR_MAXIMUM_ELONGATION  0.8

/*****************************************************************************
 Main.
 *****************************************************************************/
void main(void)
  {
  MIL_ID MilApplication,                  /* Application identifier.   */
         MilSystem,                       /* System Identifier.        */
         MilImage,                        /* Image buffer identifier.  */
         MilEdgeContext,                  /* Edge context              */
         MilResult;                       /* Result identifier.        */

  long   NumResults   = 0L,               /* Number of results found.  */
         NumEdgeFound = 0L;               /* Number of edges found.    */
  double MeanFeretDiameter[CONTOUR_MAX_RESULTS]; /* Edge mean Feret diameter. */
  int    i;                               /* Loop variable             */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, M_NULL, M_NULL, M_NULL);

  /* Load Restore the image and display it */
  MbufRestore(CONTOUR_IMAGE, MilSystem, &MilImage);

  /* Allocate a edge finder context. */
  MedgeAlloc(MilSystem, M_CONTOUR, M_DEFAULT, &MilEdgeContext);

  /* Allocate a result buffer. */
  MedgeAllocResult(MilSystem, M_DEFAULT, &MilResult);

  /* Enable features to compute */
  MedgeControl(MilEdgeContext, M_FERET_MEAN_DIAMETER+M_SORT1_DOWN, M_ENABLE);
  MedgeControl(MilEdgeContext, M_MOMENT_ELONGATION,              M_ENABLE);

  /* Calculate edges */
  MedgeCalculate(MilEdgeContext, MilImage, M_NULL, M_NULL, M_NULL, MilResult, M_DEFAULT);

  /* Get the number of found edges. */
  MedgeGetResult(MilResult, M_DEFAULT, M_NUMBER_OF_CHAINS+M_TYPE_LONG, &NumEdgeFound, M_NULL);

  /* Exclude elongated edges */
  MedgeSelect(MilResult, M_EXCLUDE, M_MOMENT_ELONGATION, M_LESS, CONTOUR_MAXIMUM_ELONGATION, M_NULL);

  /* Get the number of edges found. */
  MedgeGetResult(MilResult, M_DEFAULT, M_NUMBER_OF_CHAINS+M_TYPE_LONG, &NumResults, M_NULL);
```

Edge Finder (continued)

```
/* If edges have been found */
if ( (NumResults >= 1) && (NumResults <= CONTOUR_MAX_RESULTS) )
  {
  /* Exclude inner chains */
  MedgeSelect(MilResult, M_EXCLUDE, M_INCLUDED_EDGES, M_INSIDE_BOX, M_NULL, M_NULL);

  /* Get the number of edges found. */
  MedgeGetResult(MilResult, M_DEFAULT, M_NUMBER_OF_CHAINS+M_TYPE_LONG, &NumResults, M_NULL);

  /* Get the mean Feret diameters of the outer edges */
  MedgeGetResult(MilResult, M_DEFAULT, M_FERET_MEAN_DIAMETER, &MeanFeretDiameter, M_NULL);

  /* Now print the mean diameter of each outer edge. */
  printf("The mean diameter for each outer edge is:\n\n");
  printf("Index   Mean diameter \n");
  for (i=0; i<NumResults; i++)
    {
    printf("%-11d%-13.2f%\n", i, MeanFeretDiameter[i]);
    }
  }

/* Wait for a key press. */
printf("Press <Enter> to end.\n");
getch();

/* Free MIL objects. */
MbufFree(MilImage);
MedgeFree(MilEdgeContext);
MedgeFree(MilResult);

/* Free defaults. */
MappFreeDefault(MilApplication, MilSystem, M_NULL, M_NULL, M_NULL);
}
```

Geometric Model Finder (MIL example)

This program defines a sngle model and searches for the model in a target image based on geometric features.

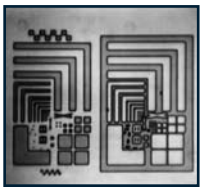


"SimpleModel.mim"

```
#include <stdio.h>
#include <mil.h>

#define MODEL_IMAGE                 "SingleModel.mim"
#define MODEL_TARGET_IMAGE          "SimpleTarget.mim"
#define MODEL_SEARCH_SPEED          M_VERY_HIGH
#define MODEL_OFFSETX               176L
#define MODEL_OFFSETY               136L
#define MODEL_SIZEX                 128L
#define MODEL_SIZEY                 128L
#define MODEL_MAX_OCCURRENCES       16L
#define MODEL_DRAW_COLOR            M_RGB888(255, 0, 0) /* Red */

void main(void)
  {
  MIL_ID MilApplication,                    /* Application identifier. */
         MilSystem,                         /* System Identifier. */
         MilDisplay,                        /* Display identifier. */
         MilImage,                          /* Image buffer identifier. */
         MilOverlayImage,                   /* Overlay image. */
         MilSearchContext,                  /* Search context */
         MilResult;                         /* Result identifier. */

  long    Model[MODEL_MAX_OCCURRENCES],     /* Model index. */
         ModelDrawColor = MODEL_DRAW_COLOR; /* Model draw color */

  long    TransparentColor,                 /* Overlay clear color. */
         NumResults  = 0L;                  /* Number of results found. */

  double Score[MODEL_MAX_OCCURRENCES],      /* Model correlation score. */
         XPosition[MODEL_MAX_OCCURRENCES],  /* Model X position. */
         YPosition[MODEL_MAX_OCCURRENCES],  /* Model Y position. */
         Angle[MODEL_MAX_OCCURRENCES],      /* Model occurrence angle. */
         Scale[MODEL_MAX_OCCURRENCES],      /* Model occurrence scale. */
         Time = 0.0;                        /* Bench variable. */

  int      i;                               /* Loop variable */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, M_NULL);

  /* Load Restore the model image and display it */
  MbufRestore(MODEL_IMAGE, MilSystem, &MilImage);
  MdispSelect(MilDisplay, MilImage);

  /* Prepare for overlay annotations. */
  MdispControl(MilDisplay, M_WINDOW_OVR_WRITE, M_ENABLE);
  MdispInquire(MilDisplay, M_WINDOW_OVR_BUF_ID, &MilOverlayImage);
  MdispInquire(MilDisplay, M_KEY_COLOR, &TransparentColor);
  if (MbufInquire(MilOverlayImage, M_SIZE_BAND, M_NULL) == 1)
     ModelDrawColor = 0xFF;

  /* Allocate a geometric model finder. */
  MmodAlloc(MilSystem, M_GEOMETRIC, M_DEFAULT, &MilSearchContext);

  /* Allocate a result buffer. */
  MmodAllocResult(MilSystem, M_DEFAULT, &MilResult);

  /* Define the model */
  MmodDefine(MilSearchContext, M_IMAGE, MilImage,
         MODEL_OFFSETX, MODEL_OFFSETY, MODEL_SIZEX, MODEL_SIZEY);

  /* Set the search speed */
  MmodControl(MilSearchContext, M_CONTEXT, M_SPEED, MODEL_SEARCH_SPEED);
  /* Preprocess the search context. */
  MmodPreprocess(MilSearchContext, M_DEFAULT);
```

```
/* Draw box and position in the source image to show the model. */
MgraColor(M_DEFAULT, ModelDrawColor);
MmodDraw(M_DEFAULT, MilSearchContext, MilOverlayImage,
        M_DRAW_BOX+M_DRAW_POSITION, 0, M_ORIGINAL);

/* Pause to show the model. */
printf("A model finder context was defined with the model in the displayed image.\n");
printf("Press <Enter> to continue.\n");
getchar();

/* Clear the overlay image. */
MbufClear(MilOverlayImage, TransparentColor);

/* Load the single model target image. */
MbufLoad(MODEL_TARGET_IMAGE, MilImage);

/* Dummy first find for better function timing accuracy (model cache effet,...). */
MmodFind(MilSearchContext, MilImage, MilResult);

/* Search for the model. */
MappTimer(M_TIMER_RESET, M_NULL);
MmodFind(MilSearchContext, MilImage, MilResult);
MappTimer(M_TIMER_READ, &Time);

/* Get the number of models found. */
MmodGetResult(MilResult, M_DEFAULT, M_NUMBER+M_TYPE_LONG, &NumResults);

/* If a model was found above the acceptance threshold. */
if ( (NumResults >= 1) && (NumResults <= MODEL_MAX_OCCURRENCES) )
  {
  /* Get the results for each model. */
  MmodGetResult(MilResult, M_DEFAULT, M_INDEX+M_TYPE_LONG, Model);
  MmodGetResult(MilResult, M_DEFAULT, M_POSITION_X, XPosition);
  MmodGetResult(MilResult, M_DEFAULT, M_POSITION_Y, YPosition);
  MmodGetResult(MilResult, M_DEFAULT, M_ANGLE, Angle);
  MmodGetResult(MilResult, M_DEFAULT, M_SCALE, Scale);
  MmodGetResult(MilResult, M_DEFAULT, M_SCORE, Score);

  /* Print the results for each model found. */
  printf("The model finder context was used to find the model in the target image.\n\n");
  printf("Result  Model  X Position  Y Position  Angle  Scale  Score\n\n");
  for (i=0; i<NumResults; i++)
    {
    printf("%-9d%-8d%-13.2f%-13.2f%-8.2f%-8.2f%-5.2f%%\n",
         i, Model[i], XPosition[i], YPosition[i], Angle[i], Scale[i], Score[i]);
    }
  printf("\nThe search time is %.1f ms\n\n", Time*1000.0);

  /* Draw edges, position and box over the occurrences that were found. */
  for (i=0; i<NumResults; i++)
    {
    MgraColor(M_DEFAULT, ModelDrawColor);
    MmodDraw(M_DEFAULT,  MilResult, MilOverlayImage,
           M_DRAW_EDGES+M_DRAW_BOX+M_DRAW_POSITION, i, M_DEFAULT);
    }
  }
else
  {
  printf("The model was not found or the number of models found is greater than\n");
  printf("the specified maximum number of occurrence !\n\n");
  }

/* Wait for a key press. */
printf("Press <Enter>.\n");
getchar();

/* Free MIL objects. */
MbufFree(MilImage);
MmodFree(MilSearchContext);
MmodFree(MilResult);

/* Free defaults. */
MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, M_NULL);
}
```

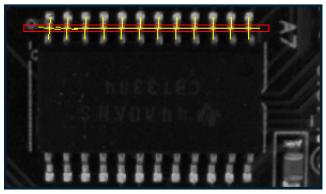
Image processing – convolution (MIL example)
This program loads an image and performs a 3x3 custom convolution operation (smoothing) on it.


"wafer.mim"

```
#include <stdio.h>
#include <mil.h>

/* Target MIL image file specifications. */
#define IMAGE_FILE                      "wafer.mim"
#define IMAGE_WIDTH                     512L
#define IMAGE_HEIGHT                    480L

/* Kernel information. */
#define KERNEL_WIDTH                    3L
#define KERNEL_HEIGHT                   3L
#define KERNEL_DEPTH                    8L

/* Average kernel information data definition. */
unsigned char   KernelData[KERNEL_HEIGHT] [KERNEL_WIDTH] =
                { {1, 2, 1},
                {2, 4, 2},
                {1, 2, 1}
                };
void main(void)
{
  MIL_ID  MilApplication,               /* Application identifier. */
          MilSystem,                    /* System identifier. */
          MilDisplay,                   /* Display identifier. */
          MilImage,                     /* Image buffer identifier. */
          MilSubImage,                  /* Sub-image buffer identifier. */
          MilKernel;                    /* Custom kernel identifier. */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, &MilImage);

  /* Restrict the region to be processed to the image size. */
  MbufChild2d(MilImage, 0L, 0L, IMAGE_WIDTH, IMAGE_HEIGHT, &MilSubImage);

  /* Load source image into an image buffer. */
  MbufLoad(IMAGE_FILE, MilSubImage);

  /* Pause to show the original image. */
  printf("This program does a convolution on the displayed image.\n");
  printf("It uses a custom smoothing kernel.\n");
  printf("Press <Enter> to continue.");
  getchar();

  /* Allocate a MIL kernel. */
  MbufAlloc2d(M_DEFAULT, KERNEL_HEIGHT, KERNEL_WIDTH, KERNEL_DEPTH+M_UNSIGNED,  M_KERNEL, &MilKernel);

  /* Put the custom data in it. */
  MbufPut(MilKernel, KernelData);

  /* Set a normalization (divide) factor to have a kernel with a sum equal to one. */
  MbufControlNeighborhood(MilKernel, M_NORMALIZATION_FACTOR, 16L);

  /* Convolve the image using the kernel. */
  MimConvolve(MilSubImage, MilSubImage, MilKernel);

  /* Pause to show the result. */
  printf("\n");
  printf("The original image was smoothed using a custom kernel.\n");
  printf("Press <Enter> to terminate.");
  getchar();

  /* Free all allocations. */
  MbufFree(MilKernel);
  MbufFree(MilSubImage);
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, MilImage);
}
```

Measurement (MIL example)

This program measures the position, width and angle of a stripe in an image, and marks its center and edges.



"chip.mim"

```
/* Regular includes. */
#include <stdio.h>
#include <mil.h>

/* Source MIL image file specification. */
#define IMAGE_FILE                      "chip.mim"

/* Processing region specification. */
#define MEAS_BOX_WIDTH                  128
#define MEAS_BOX_HEIGHT                 100
#define MEAS_BOX _POS_X                 166
#define MEAS_BOX _POS_Y                 130

/* Target stripe typical specifications. */
#define STRIPE_POLARITY_LEFT            M_POSITIVE
#define STRIPE_POLARITY_RIGHT           M_NEGATIVE
#define STRIPE_WIDTH                    45L
#define STRIPE_WIDTH_VARIATION          10L

/* Size and color of the cross to mark the positions. */
#define CROSS_SIZE                      10L
#define CROSS_COLOR                     240L

/* Utility functions prototypes */
void DrawCross(MIL_ID ImageId, double CenterX, double CenterY, long Color);

/* Main application function */
void main(void)
{
   MIL_ID MilApplication,               /* Application identifier */
          MilSystem,                    /* System identifier. */
          MilDisplay,                   /* Display identifier. */
          MilImage,                     /* Image buffer identifier. */
          StripeMarker;                 /* Stripe marker identifier. */
   double  StripeCenterX                /* Stripe X center position. */
          StripeCenterY                 /* Stripe Y center position. */
          StripeWidth,                  /* Stripe width. */
          StripeAngle                   /* Stripe angle. */
          StripeScore,                  /* Stripe Score. */
          StripeFirstEdgeX,             /* Stripe left edge X position. */
          StripeFirstEdgeY,             /* Stripe left edge Y position. */
          StripeSecondEdgeX,            /* Stripe right edge X position. */
          StripeSecondEdgeY;            /* Stripe right edge Y position. */

   /* Allocate defaults */
    MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, &MilImage);

   /* Read the source image */
    MbufLoad(IMAGE_FILE, MilImage);

   /* Draw the contour of the measurement box */
    MgraRect(M_DEFAULT, MilImage, MEAS_BOX_POS_X-1, MEAS_BOX_POS_Y-1,
             MEAS_BOX_POS_X+MEAS_BOX_WIDTH+1, MEAS_BOX_POS_Y+MEAS_BOX_HEIGHT+1);

   /* Pause to show the original image. */
    printf("This program will determine the position, width and angle of the\n");
    printf("stripe in the highlighted box and mark its center and edges.\n");
    printf("Press <Enter> to continue.\n\n");
    getchar();
```

Measurement (continued)

```
/* Read the source image again to remove previously drawn rectangle */
   MbufLoad(IMAGE_FILE, MilImage);
/* Allocate a stripe marker */
 MmeasAllocMarker(M_DEFAULT, M_STRIPE, M_DEFAULT, &StripeMarker);

/* Specify the stripe approximative definition */
 MmeasSetMarker(StripeMarker, M_POLARITY, STRIPE_POLARITY_LEFT, STRIPE_POLARITY_RIGHT);
 MmeasSetMarker(StripeMarker, M_WIDTH, STRIPE_WIDTH, M_NULL);
 MmeasSetMarker(StripeMarker, M_WIDTH_VARIATION, STRIPE_WIDTH_VARIATION, M_NULL);
 MmeasSetMarker(StripeMarker, M_BOX_ANGLE_MODE, M_ENABLE, M_NULL);

/* Specify the search box size. */
 MmeasSetMarker(StripeMarker, M_BOX_ORIGIN, MEAS_BOX_POS_X, MEAS_BOX_POS_Y);
 MmeasSetMarker(StripeMarker, M_BOX_SIZE, MEAS_BOX_WIDTH, MEAS_BOX_HEIGHT);

/* Find the stripe and measure its width and angle. */
 MmeasFindMarker(M_DEFAULT, MilImage , StripeMarker, M_DEFAULT);

/* Get the stripe position, width and angle. */
 MmeasGetResult(StripeMarker, M_POSITION, &StripeCenterX, &StripeCenterY);
 MmeasGetResult(StripeMarker, M_POSITION+M_EDGE_FIRST, &StripeFirstEdgeX, &StripeFirstEdgeY);
 MmeasGetResult(StripeMarker, M_POSITION+M_EDGE_SECOND, &StripeSecondEdgeX, &StripeSecondEdgeY);
 MmeasGetResult(StripeMarker, M_WIDTH, &StripeWidth, M_NULL);
 MmeasGetResult(StripeMarker, M_ANGLE, &StripeAngle, M_NULL);
 MmeasGetResult(StripeMarker, M_SCORE, &StripeScore, M_NULL);

/* Draw a cross on the center, left edge and right edge of the found stripe. */
 DrawCross(MilImage, StripeCenterX, StripeCenterY, CROSS_COLOR);
 DrawCross(MilImage, StripeFirstEdgeX, StripeFirstEdgeY, CROSS_COLOR);
 DrawCross(MilImage, StripeSecondEdgeX, StripeSecondEdgeY, CROSS_COLOR);

/* Draw the contour of the measurement box */
 MgraRect(M_DEFAULT, MilImage, MEAS_BOX_POS_X-1, MEAS_BOX_POS_Y-1, MEAS_BOX_POS_X+MEAS_BOX_WIDTH+1,
          MEAS_BOX_POS_Y+MEAS_BOX_HEIGHT+1);

/* Print the result. */
 printf("The stripe in the box is at position %.2f,%.2f and\n", StripeCenterX, StripeCenterY);
 printf("is %.2f pixels wide with an angle of %.2f degrees.\n", StripeWidth, StripeAngle);
 printf("Its center and edges have been marked.\n");
 printf("Press <Enter> to continue.\n\n");
 getchar();

/* Free all allocations. */
 MmeasFree(StripeMarker);
 MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, MilImage);
}

/* Draw a cross at the specified position. */
void DrawCross(MIL_ID ImageId, double CenterX, double CenterY, long Color)

{
   MgraColor(M_DEFAULT,Color);
   MgraLine(M_DEFAULT, ImageId,(long)(CenterX+.5)-(CROSS_SIZE/2), (long)(CenterY+.5),
            (long)(CenterX+.5)+(CROSS_SIZE/2), (long)(CenterY+.5));
   MgraLine(M_DEFAULT, ImageId, (long)(CenterX+.5), (long)(CenterY+.5)-(CROSS_SIZE/2), (long)(CenterX+.5),
            (long)(CenterY+.5)+(CROSS_SIZE/2));
}
```

Multi-buffered image capture and processing (MIL example)

This program shows the use of the MdigProcess() function to perform real-time image capture and processing.

```
#include <mil.h>
#include <conio.h>
#include <stdlib.h>

 /* Number of images in the buffering grab queue.    Generally, increasing this number gives better real-time grab.  */
#define BUFFERING_SIZE_MAX 20

/* User's processing function prototype. */
long MFTYPE ProcessingFunction(long HookType, MIL_ID HookId, void MPTYPE *HookDataPtr);

/* User's processing function hook data structure. */
typedef struct
  {
  MIL_ID  MilImageDisp;
  long    ProcessedImageCount;
  } HookDataStruct;

/* Main function. */
/* ---------------*/

void main(void)
{
  MIL_ID MilApplication;
  MIL_ID MilSystem     ;
  MIL_ID MilDigitizer  ;
  MIL_ID MilDisplay    ;
  MIL_ID MilImageDisp  ;
  MIL_ID MilGrabBufferList[BUFFERING_SIZE_MAX] = { 0 };
  long   MilGrabBufferListSize;
  long   ProcessFrameCount  = 0;
  long   NbFrames           = 0;
  double ProcessFrameRate   = 0;
  HookDataStruct UserHookData;

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay,
                                  &MilDigitizer, &MilImageDisp);

  /* Allocate the grab buffers and clear them. */
  MappControl(M_ERROR, M_PRINT_DISABLE);
  for(MilGrabBufferListSize = 0; MilGrabBufferListSize<BUFFERING_SIZE_MAX; MilGrabBufferListSize++)
    {
    MbufAlloc2d(MilSystem,
             MdigInquire(MilDigitizer, M_SIZE_X, M_NULL),
             MdigInquire(MilDigitizer, M_SIZE_Y, M_NULL),
             M_DEF_IMAGE_TYPE,
             M_IMAGE+M_GRAB+M_PROC,
             &MilGrabBufferList[MilGrabBufferListSize]);

    if (MilGrabBufferList[MilGrabBufferListSize])
      {
      MbufClear(MilGrabBufferList[MilGrabBufferListSize], 0xFF);
      }
    else
      break;
    }
```

Multi-buffered image capture and processing (continued)

```
  MappControl(M_ERROR, M_PRINT_ENABLE);
  /* Free a buffer to leave space for possible temporary buffer. */
  MilGrabBufferListSize--;
  MbufFree(MilGrabBufferList[MilGrabBufferListSize]);

  /* Print a message. */
  printf("\nMULTIPLE BUFFERED PROCESSING.\n");
  printf("----------------------------\n\n");
  printf("Press <Enter> to start.\n\n");

  /* Grab continuously on the display and wait for a key press. */
  MdigGrabContinuous(MilDigitizer, MilImageDisp);
  getch();

  /* Halt continuous grab. */
  MdigHalt(MilDigitizer);

  /* Initialize the User's processing function data structure. */
  UserHookData.MilImageDisp       = MilImageDisp;
  UserHookData.ProcessedImageCount = 0;

  /* Start the processing. The processing function is called for every frame grabbed. */
  MdigProcess(MilDigitizer, MilGrabBufferList, MilGrabBufferListSize,
                      M_START, M_DEFAULT, ProcessingFunction, &UserHookData);

  /* NOTE: Now the main() is free to perform other tasks while the processing is executing. */
  /* ------------------------------------------------------------------------------------ */

  /* Print a message and wait for a key press after a minimum number of frames. */
  printf("Press <Enter> to stop.\n\n");
  getch();

  /* Stop the processing. */
  MdigProcess(MilDigitizer, MilGrabBufferList, MilGrabBufferListSize,
           M_STOP, M_DEFAULT, ProcessingFunction, &UserHookData);

  /* Print statistics. */
  MdigInquire(MilDigitizer, M_PROCESS_FRAME_COUNT,  &ProcessFrameCount);
  MdigInquire(MilDigitizer, M_PROCESS_FRAME_RATE,   &ProcessFrameRate);
  printf("\n\n%ld frames grabbed at %.1f frames/sec (%.1f ms/frame).\n",
                      ProcessFrameCount, ProcessFrameRate, 1000.0/ProcessFrameRate);
  printf("Press <Enter> to end.\n\n");
  getch();

  /* Free the grab buffers. */
  while(MilGrabBufferListSize > 0)
    MbufFree(MilGrabBufferList[--MilGrabBufferListSize]);

  /* Release defaults. */
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, MilDigitizer, MilImageDisp);
}

/* User's processing function called every time a grab buffer is modified. */
/* ------------------------------------------------------------------------*/
```

Multi-buffered image capture and processing (continued)

```
/* Local defines. */
#define STRING_LENGTH_MAX  20
#define STRING_POS_X       20
#define STRING_POS_Y       20
long MFTYPE ProcessingFunction(long HookType, MIL_ID HookId, void MPTYPE *HookDataPtr)
  {
  HookDataStruct *UserHookDataPtr = (HookDataStruct *)HookDataPtr;
  MIL_ID ModifiedBufferId;
  MIL_TEXT_CHAR Text[STRING_LENGTH_MAX]= {'\0',};

  /* Retrieve the MIL_ID of the grabbed buffer. */
  MdigGetHookInfo(HookId, M_MODIFIED_BUFFER+M_BUFFER_ID, &ModifiedBufferId);

  /* Print and draw the frame count. */
  UserHookDataPtr->ProcessedImageCount++;
  printf("Processing frame #%d.\r", UserHookDataPtr->ProcessedImageCount);
  MOs_ltoa(UserHookDataPtr->ProcessedImageCount, Text, 10);
  MgraText(M_DEFAULT, ModifiedBufferId, STRING_POS_X, STRING_POS_Y, Text);

  /* Perform the processing and update the display. */
  #if (!M_MIL_LITE)
    MimArith(ModifiedBufferId, M_NULL, UserHookDataPtr->MilImageDisp, M_NOT);
  #else
    MbufCopy(ModifiedBufferId, UserHookDataPtr->MilImageDisp);
  #endif

  return 0;
  }
```

OCR (MIL example)

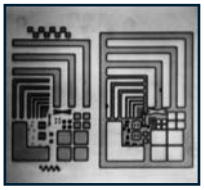
This program calibrates an OCR font (semi-font) and uses it to read a string present in the image. The string is then printed to the screen and the calibrated font is saved to disk.

"ocrsemi1.mim"

```
#include <stdio.h>
#include <string.h>
#include <mil.h>

/* Target image character specifications. */
#define CHAR_IMAGE_FILE          "ocrsemi1.mim"
#define CHAR_SIZE_X_MIN          22.0
#define CHAR_SIZE_X_MAX          23.0
#define CHAR_SIZE_X_STEP         0.50
#define CHAR_SIZE_Y_MIN          43.0
#define CHAR_SIZE_Y_MAX          44.0
#define CHAR_SIZE_Y_STEP         0.50

/* Target reading specifications. */
#define READ_REGION_POS_X        30L
#define READ_REGION_POS_Y        40L
#define READ_REGION_WIDTH        420L
#define READ_REGION_HEIGHT       70L
#define READ_SCORE_MIN           50.0

/* Font file names. */
#define FONT_FILE_IN             "semi1292.mfo"
#define FONT_FILE_OUT            "semicali.mfo"

/* Length of the string to read (null terminated) */
#define STRING_LENGTH            13L
#define STRING_CALIBRATION       "M940902-MXD5"

/* Drawing color for the resulting string */
#define STRING_DRAWING_COLOR     255L

void main(void)
{
 MIL_ID MilApplication,          /* Application identifier. */
        MilSystem,               /* System identifier. */
        MilDisplay,              /* Display identifier. */
        MilImage,                /* Image buffer identifier. */
        MilSubImage,             /* Sub-image buffer identifier. */
        OcrFont,                 /* OCR font identifier. */
        OcrResult;               /* OCR result buffer identifier. */
 char    String[STRING_LENGTH];  /* Array of characters to read. */
 double  Score;                  /* Reading score. */

 /* Allocate defaults */
 MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, &MilImage);

 /* Load source image into image buffer. */
 MbufLoad(CHAR_IMAGE_FILE, MilImage);

 /* Restrict the region of the image where to read the string.*/
 MbufChild2d(MilImage, READ_REGION_POS_X, READ_REGION_POS_Y, READ_REGION_WIDTH,
             READ_REGION_HEIGHT, &MilSubImage);
 /* Restore the OCR character font from disk. */
 MocrRestoreFont(FONT_FILE_IN, M_RESTORE, MilSystem, &OcrFont);

/* Pause to show the original image and ask the calibration string. */
 printf("The OCR font will be calibrated using the displayed image.\n");
 printf("\nCalibrating font...\n\n");
```

```
/* Calibrate the OCR font. */
 MocrCalibrateFont(MilSubImage, OcrFont, STRING_CALIBRATION, CHAR_SIZE_X_MIN, CHAR_SIZE_X_MAX,
                   CHAR_SIZE_X_STEP, CHAR_SIZE_Y_MIN, CHAR_SIZE_Y_MAX, CHAR_SIZE_Y_STEP, M_DEFAULT);

 /* Set the user specific character constraints for each string position */
 MocrSetConstraint(OcrFont, 0,  M_LETTER, M_NULL);                    /*  A to Z only */
 MocrSetConstraint(OcrFont, 1,  M_DIGIT,  "9");                       /*  9 only */
 MocrSetConstraint(OcrFont, 2,  M_DIGIT,  M_NULL);                    /*  0 to 9 only */
 MocrSetConstraint(OcrFont, 3,  M_DIGIT,  M_NULL);                    /*  0 to 9 only */
 MocrSetConstraint(OcrFont, 4,  M_DIGIT,  M_NULL);                    /*  0 to 9 only */
 MocrSetConstraint(OcrFont, 5,  M_DIGIT,  M_NULL);                    /*  0 to 9 only */
 MocrSetConstraint(OcrFont, 6,  M_DIGIT,  M_NULL);                    /*  0 to 9 only */
 MocrSetConstraint(OcrFont, 7,  M_DEFAULT,"-");                       /*  -  only */
 MocrSetConstraint(OcrFont, 8,  M_LETTER, "M");                       /*  M only */
 MocrSetConstraint(OcrFont, 9,  M_LETTER, "X");                       /*  X only */
 MocrSetConstraint(OcrFont, 10, M_LETTER, "ABCDEFGH");                /*  SEMI checksum */
 MocrSetConstraint(OcrFont, 11, M_DIGIT,  "01234567");                /*  SEMI checksum            */

 /* Pause to signal the following read operation. */
 printf("The string present in the displayed image will be read and\n");
 printf("the result will be printed.\nPress <Enter> to continue.\n");
 getchar();

 /* Allocate an OCR result buffer. */
 MocrAllocResult(MilSystem, M_DEFAULT, &OcrResult);

 /* Read the string. */
 MocrReadString(MilSubImage, OcrFont, OcrResult);


 /* Get the string and its reading score. */
 MocrGetResult(OcrResult, M_STRING, String);
 MocrGetResult(OcrResult, M_SCORE, &Score);

 /* Print the result. */
 printf("\nThe string read is: \"%s\" (score: %.1f%%).\n\n", String, Score);

 /* Draw the string under the reading region. */
 MgraFont(M_DEFAULT, M_FONT_DEFAULT_LARGE);
 MgraColor(M_DEFAULT,STRING_DRAWING_COLOR);
 MgraText(M_DEFAULT, MilImage, READ_REGION_POS_X+(READ_REGION_WIDTH/4),
          READ_REGION_POS_Y+READ_REGION_HEIGHT, String);

 /* Save the calibrated font if the reading score was sufficient. */
 if (Score > READ_SCORE_MIN)
 {
    MocrSaveFont(FONT_FILE_OUT, M_SAVE, OcrFont);
    printf("Read successful, calibrated OCR font was saved to disk.\n");
 }
else
 {
    printf("Error: Read score too low, calibrated OCR font not saved.\n");
 }

 printf("Press <Enter> to end.\n");
 getchar();

 /* Free all allocations. */
 MocrFree(OcrFont);
 MocrFree(OcrResult);
 MbufFree(MilSubImage);
 MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, MilImage);
}
```

Pattern matching (MIL example)
This program finds the horizontal and vertical displacement of a wafer image.


"wafer.mim"

```
#include <stdio.h>

#include <mil.h>

/* Source and target images file specifications. */
#define MODEL_IMAGE_FILE                "wafer.mim"
#define TARGET_IMAGE_FILE               "shfwafer.mim"
#define IMAGE_WIDTH                     512L
#define IMAGE_HEIGHT                    480L

/* Model width, height, maximum displacement, initial position */
#define MODEL_WIDTH                     64L
#define MODEL_HEIGHT                    64L

void main(void)
{

 MIL_ID MilApplication,                 /* Application identifier. */
        MilSystem,                      /* System identifier. */
        MilDisplay,                     /* Display identifier.  */
        MilImage,                       /* Image buffer identifier. */
        MilSubImage,                    /* Sub-image buffer identifier. */
        Model,                          /* Model identifier. */
        Result;                         /* Result buffer identifier. */
long    PosX, PosY;                     /* Model position. */
long    AllocError;                     /* Allocation error variable. */
double  OrgX=0.0, OrgY=0.0;             /* Original center of model. */
double  x=0.0, y=0.0, Score=0.0;        /* Result variables. */

/* Allocate defaults. */
MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, &MilImage);

/* Load model image into an image buffer. */
MbufLoad(MODEL_IMAGE_FILE, MilImage);

/* Restrict the region to be processed to the bottom right corner of the image. */
MbufChild2d(MilImage, IMAGE_WIDTH/2, IMAGE_HEIGHT/2, IMAGE_WIDTH/2, IMAGE_HEIGHT/2, &MilSubImage);

/* Announce the automatic model definition. */
printf("A model is being automatically defined in the source image, ");
printf("please wait...\n\n");

/* Automatically allocate normalized grayscale type model. */
MpatAllocAutoModel(MilSystem, MilSubImage, MODEL_WIDTH, MODEL_HEIGHT, M_DEFAULT,
                   M_DEFAULT, M_NORMALIZED, M_DEFAULT, &Model);

/* Check for a successful model allocation. */
MappGetError(M_CURRENT, &AllocError);
if (!AllocError)
  {
  MpatInquire(Model,M_ALLOC_OFFSET_X+M_TYPE_LONG,&PosX);
  MpatInquire(Model,M_ALLOC_OFFSET_Y+M_TYPE_LONG,&PosY);
  MpatInquire(Model,M_ORIGINAL_X,&OrgX);
  MpatInquire(Model,M_ORIGINAL_Y,&OrgY);
```

Pattern matching (continued)

```
   /* Draw box around model. */
   MgraRect(M_DEFAULT, MilSubImage, PosX - 1, PosY - 1, PosX + MODEL_WIDTH, PosY + MODEL_HEIGHT);
   printf("Model successfully defined as shown on the displayed image.\n");
   printf("Press <Enter> to continue.\n");
   getchar();

   /* Load target image into an image buffer. */
   MbufLoad(TARGET_IMAGE_FILE, MilImage);

   /* Allocate result. */
   MpatAllocResult(MilSystem, 1L, &Result);

   /* Find model. */
   MpatFindModel(MilSubImage, Model, Result);

   /* If one model was found above the acceptance threshold set. */
   if (MpatGetNumber(Result, M_NULL) == 1L)
     {
     /* Get results. */
     MpatGetResult(Result, M_POSITION_X, &x);
     MpatGetResult(Result, M_POSITION_Y, &y);
     MpatGetResult(Result, M_SCORE, &Score);

     /* Draw a box around occurrence. */
     MgraRect(M_DEFAULT, MilSubImage,
           (long)(x + 0.5) - (MODEL_WIDTH/2)   - 1,
           (long)(y + 0.5) - (MODEL_HEIGHT/2) - 1,
           (long)(x + 0.5) + (MODEL_WIDTH/2),
           (long)(y + 0.5) + (MODEL_HEIGHT/2));

     /* Analyze and print results. */
     printf("A misaligned version of the source image was loaded.\n\n");
     printf("Image was found to be offset by %.2f in X, and %.2f in Y.\n", x - OrgX, y - OrgY);
     printf("Model match score is %.1f percent.\n", Score);
     printf("Press <Enter> to end.\n");
     getchar();
     }
   else
     {
     printf("Error: Pattern not found properly.\n");
     printf("Press <Enter> to end.\n");
     getchar();
     }

   /* Free result buffer and model. */
   MpatFree(Result);
   MpatFree(Model);
   }
    else
   {
   printf("Error: Automatic model definition failed.\n");
   printf("Press <Enter> to end.\n");
   getchar();
   }

/* Free child image and defaults. */
MbufFree(MilSubImage);
MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, MilImage);
}
```

String Reader (MIL example)

This program uses the String Reader module to define a font from an image containing a mosaic of license plates. Two string models are then defined and parameterized to read only valid license plates. License plate reading is then performed in a target image of a car on a road.

```
#include <mil.h>
#include <conio.h>

/* MIL image file specifications. */
#define IMAGE_FILE_DEFINITION M_IMAGE_PATH MIL_TEXT("QcPlates.mim")
#define IMAGE_FILE_TO_READ    M_IMAGE_PATH MIL_TEXT("LicPlate.mim")

/* String containing all characters used for font definition. */
#define TEXT_DEFINITION       "AVS300CVK781JNK278 EBX380QKN918HCC839 YRH900ZQR756977AMQ GPK742389EYE569ESQ"

/* Font normalization size Y. */
#define NORMALIZATION_SIZE_Y    20L

/* Max size of plate string. */
#define STRING_MAX_SIZE         32L
```

"QcPlates.mim"

```
/*****************************************************************************/
/* Main. */
void main(void)
  {
  MIL_ID MilApplication,                  /* Application identifier.          */
         MilSystem,                       /* System identifier.               */
         MilDisplay,                      /* Display identifier.              */
         MilImage,                        /* Image buffer identifier.         */
         MilOverlayImage,                 /* Overlay image.                   */
         MilStrContext,                   /* String context identifier.       */
         MilStrResult;                    /* String result buffer identifier. */
  long   NumberOfStringRead;              /* Total number of strings to read. */
  double Score;                           /* String score.                    */
  char*  StringResult[STRING_MAX_SIZE+1];  /* String of characters read.       */
  double Time = 0.0;                      /* Time variable.                   */

  /* Print module name. */
  printf("\nSTRING READER MODULE:\n");
  printf("---------------------\n\n");

  /* Allocate defaults */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, M_NULL);

  /* Restore the font definition image */
  MbufRestore(IMAGE_FILE_DEFINITION, MilSystem, &MilImage);

  /* Display the image and prepare for overlay annotations. */
  MdispSelect(MilDisplay, MilImage);
  MdispControl(MilDisplay, M_OVERLAY, M_ENABLE);
  MdispInquire(MilDisplay, M_OVERLAY_ID, &MilOverlayImage);

  /* Allocate a new empty String Reader context. */
  MstrAlloc( MilSystem, M_FEATURE_BASED, M_DEFAULT, &MilStrContext);

  /* Allocate a new empty String Reader result buffer. */
  MstrAllocResult(MilSystem, M_DEFAULT, &MilStrResult);

  /* Add a new empty user defined font to the context. */
  MstrControl(MilStrContext, M_CONTEXT, M_FONT_ADD, M_USER_DEFINED);
```

"LicPlate.mim"

String Reader (continued)

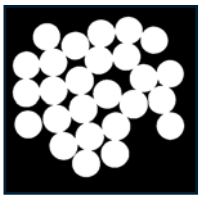
```
/* Add user defined characters from the license plate mosaic image. */
MstrEditFont(MilStrContext, M_FONT_INDEX(0), M_CHAR_ADD,
          M_USER_DEFINED + M_FOREGROUND_BLACK,
          MilImage, TEXT_DEFINITION, M_NULL);

/* Draw all the characters in the font in the overlay image. */
MgraColor(M_DEFAULT, M_COLOR_GREEN);
MstrDraw(M_DEFAULT, MilStrContext, MilOverlayImage, M_DRAW_CHAR,
       M_FONT_INDEX(0), M_NULL, M_ORIGINAL);

/* Normalize the characters of the font to an appropriate size. */
MstrEditFont(MilStrContext, M_FONT_INDEX(0), M_CHAR_NORMALIZE,
          M_SIZE_Y, NORMALIZATION_SIZE_Y, M_NULL, M_NULL);

/* Add 2 new empty strings models to the context for the 2 valid types of
   plates (3 letters followed by 3 numbers or 3 numbers followed by 3 letters)
   Note that the read time increases with the number of strings in the context.
*/
MstrControl(MilStrContext, M_CONTEXT, M_STRING_ADD, M_USER_DEFINED);
MstrControl(MilStrContext, M_CONTEXT, M_STRING_ADD, M_USER_DEFINED);

/* Set number of strings to read. */
MstrControl(MilStrContext, M_CONTEXT, M_STRING_NUMBER, 1);

/* Set number of expected characters for all string models to 6 characters. */
MstrControl(MilStrContext, M_STRING_INDEX(M_ALL), M_STRING_SIZE_MIN, 6);
MstrControl(MilStrContext, M_STRING_INDEX(M_ALL), M_STRING_SIZE_MAX, 6);

/* Set default constraint to uppercase letter for all string models */
MstrSetConstraint(MilStrContext, M_STRING_INDEX(0), M_DEFAULT, M_LETTER + M_UPPERCASE, M_NULL );
MstrSetConstraint(MilStrContext, M_STRING_INDEX(1), M_DEFAULT, M_LETTER + M_UPPERCASE, M_NULL );

/* Set constraint of 3 last characters to digit for the first string model */
MstrSetConstraint(MilStrContext, M_STRING_INDEX(0), 3, M_DIGIT, M_NULL );
MstrSetConstraint(MilStrContext, M_STRING_INDEX(0), 4, M_DIGIT, M_NULL );
MstrSetConstraint(MilStrContext, M_STRING_INDEX(0), 5, M_DIGIT, M_NULL );

/* Set constraint of 3 first characters to digit for the second string model */
MstrSetConstraint(MilStrContext, M_STRING_INDEX(1), 0, M_DIGIT, M_NULL );
MstrSetConstraint(MilStrContext, M_STRING_INDEX(1), 1, M_DIGIT, M_NULL );
MstrSetConstraint(MilStrContext, M_STRING_INDEX(1), 2, M_DIGIT, M_NULL );

/* Pause to show the font definition. */
printf("This program has defined a font with this Quebec plates mosaic image.\n");
printf("Press <Enter> to continue.\n\n");
getch();

/* Clear the display overlay. */
MdispControl(MilDisplay, M_OVERLAY_CLEAR, M_DEFAULT);

/* Load image to read into image buffer. */
MbufLoad(IMAGE_FILE_TO_READ, MilImage);

/* Preprocess the String Reader context. */
MstrPreprocess(MilStrContext, M_DEFAULT);

/* First, perform a dummy read for better function timing accuracy (model cache effet,...). */
MstrRead(MilStrContext, MilImage, MilStrResult);

/* Reset the timer. */
MappTimer(M_TIMER_RESET+M_SYNCHRONOUS, M_NULL);
```

String Reader (continued)

```
/* Perform the read operation on the specified target image. */
MstrRead(MilStrContext, MilImage, MilStrResult);

/* Read the time. */
MappTimer(M_TIMER_READ+M_SYNCHRONOUS, &Time);

/* Get number of strings read and show the result. */
MstrGetResult(MilStrResult, M_GENERAL, M_STRING_NUMBER + M_TYPE_LONG, &NumberOfStringRead);
if( NumberOfStringRead >= 1)
  {
  printf("The license plate was read successfully (%.2lf ms)\n\n", Time*1000 );

  /* Draw read result. */
  MgraColor(M_DEFAULT, M_COLOR_BLUE);
  MstrDraw(M_DEFAULT, MilStrResult, MilOverlayImage, M_DRAW_STRING, M_ALL, M_NULL, M_DEFAULT);
  MgraColor(M_DEFAULT, M_COLOR_GREEN);
  MstrDraw(M_DEFAULT, MilStrResult, MilOverlayImage, M_DRAW_STRING_BOX, M_ALL, M_NULL, M_DEFAULT);

  /* Print the read result. */
  printf(" String                Score\n" );
  printf(" ---------------------------------------\n" );
  MstrGetResult(MilStrResult, 0, M_STRING, StringResult);
  MstrGetResult(MilStrResult, 0, M_STRING_SCORE, &Score);
  printf(" %s               %.1lf\n", StringResult, Score );
  }
else
  {
  printf("Error: Plate was not read.\n");
  }

/* Pause to show results. */
printf("\nPress <Enter> to end.\n\n");
getch();

/* Free all allocations. */
MstrFree(MilStrContext);
MstrFree(MilStrResult);
MbufFree(MilImage);

/* Free defaults. */
MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, M_NULL);
}
```

Watershed segmentation (MIL example)

This program uses watershed and distance functions to separate touching objects in a binary image.



"binpills.mim"

```
/* Regular includes. */
#include <mil.h>
#include <stdio.h>

/* Source image specifications. */
#define IMAGE_FILE        "binpills.mim"

/* Minimal distance variations for the watershed calculation. */
#define WSHED_MINIMAL_DISTANCE_VARIATION       2

/* Main application function */
void main()
{
  MIL_ID  MilApplication,            /* Application identifier. */
          MilSystem,                 /* System identifier. */
          MilDisplay,                /* Display identifier. */
          MilImage,                  /* Image buffer identifier. */
           MilImageWatershed;        /* Image buffer identifier. */

  /* Allocate defaults. */
  MappAllocDefault(M_SETUP, &MilApplication, &MilSystem, &MilDisplay, M_NULL, MilImage);

  /* Restore the source image into an automatically allocated
  * image work buffer and copy it to the display image.
  */
  MbufRestore(IMAGE_FILE, MilSystem, &MilImageWatershed);
  MbufCopy(MilImageWathershed, &MilImage);

  /* Pause to show the original image. */
  printf("Original image.\n");
  printf("Press <Enter> to continue.\n\n");
  getchar();

  /* Perform a distance transformation on the binary image. */
  MimDistance(MilImage, MilImageWatershed, M_CHAMFER_3_4);

  /* Find the watersheds of the distance image. */
  MimWatershed(MilImageWatershed, M_NULL, MilImageWatershed, WSHED_MINIMAL_DISTANCE_VARIATION,
                    M_STRAIGHT_WATERSHED + M_MAXIMA_FILL + M_SKIP_LAST_LEVEL);

  /* AND the watershed image with the original binary image
   * to separate the touching pills.
   */
  MimArith(MilImageWatershed, MilImage, MilImage, M_AND);

  /* Pause to show the segmented image. */
  printf("Segmented image.\n");
  printf("Press <Enter> to end.\n\n");
  getchar();

  /* Free all allocations */
  MbufFree(MilImageWatershed);
  MappFreeDefault(MilApplication, MilSystem, MilDisplay, M_NULL, MilImage);
}
```