IAC Vision Skills.hdev

main (: : :)

```
* Vision Skills necessary for the IAC reference programs
* IAC008 PinCount
* IAC009 PickFromTable
* IAC011 ReadID
* IAC012 FindPart (in progress)

WindowHandle1 := 99
Message_idx := 0
Continue := true
while (Continue == true)
  * Open up a framegrabber to get the desired image from the correct camera
  open_framegrabber ('USB3Vision', 0, 0, 0, 0, 0, 0, 'progressive', -1, 'default', -1,
'false', 'default', '2676014F7F1B_Basler_acA460010uc', 0, -1, AcqHandle)
  set_framegrabber_param (AcqHandle, 'OffsetY', 0)
  set_framegrabber_param (AcqHandle, 'CenterX', 0)
  set_framegrabber_param (AcqHandle, 'PixelFormat', 'Mono8')
  set_framegrabber_param (AcqHandle, 'BlackLevel', 0.0)
  get_framegrabber_param (AcqHandle, 'image_width', image_width)
  get_framegrabber_param (AcqHandle, 'image_height', image_height)

  if (WindowHandle1 == 99)
    dev_open_window (50, 1200, 1144, 818, 'black', WindowHandle1)
    set_display_font (WindowHandle1, 16, 'mono', 'true', 'false')
  endif

  * Test Routines
  while (false)
    grab_image (Image, AcqHandle)
    dev_clear_window ()
    disp_image (Image, WindowHandle1)

    CountPins (Image, SelectedRegions, row1, column1, row2, column2)
    * draw_rectangle1(WindowHandle1, 100.0, 100.0, 300.0, 300.0)
    * draw_rectangle1 (WindowHandle1, row1, column1, row2, column2)

    * Assemble and return response
    count_obj (SelectedRegions, nPins)
    Response := 'nPins = ' + nPins
    disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
  endwhile

  * Open a socket that accepts connection requests
  dev_clear_window ()
  caption := 'Waiting for client connection image=(' + image_width + ', ' + image_height +
')'
  disp_message (WindowHandle1, caption, 'window', 12, 12, 'black', 'true')
  open_socket_accept (3000, 'protocol', 'TCP4', AcceptingSocket)
```

```
  * Wait for an incoming connection
  OpenStatus := 5
  while (OpenStatus[0] != 2)
    OpenStatus := 2
    try
      socket_accept_connect (AcceptingSocket, 'false', Socket)
    catch (OpenStatus)
    endtry
    wait_seconds (0.2)
  endwhile
  dev_clear_window ()
  disp_message (WindowHandle1, 'Connected!', 'window', 12, 12, 'green', 'false')

  * Setup for intrinsic cal
  I := 1

  while (OpenStatus[0] == 2)
    try
      * Wait for a command from the controller
      receive_data (Socket, 'z', Command, From)
      Message_idx := Message_idx + 1
      * Grab image and display Command in it
      grab_image (Image, AcqHandle)
      dev_clear_window ()
      disp_image (Image, WindowHandle1)
      disp_message (WindowHandle1, 'Command: ' + Command, 'window', 12, 12, 'green',
'false')

      * Interpret command as list of comma-delimited numbers possibly followed by \n
      * Eliminate any \n
      tuple_regexp_replace (Command, '\n', '', command_clean)
      * Comma split into strings
      tuple_split (command_clean, ',', command_strings)
      * Convert strings to numbers
      tuple_number (command_strings, command_num_params)

      * Command selection is performed by the first number!
      switch (command_num_params[0])

        * Acquire only (which was done above)
      case 0:
        Response := '(' + Message_idx + ', 1)'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break

      case 2:
        * Use blob to count pins- very crude check
        CountPins (Image, SelectedRegions, row1, column1, row2, column2)
        * draw_rectangle1 (WindowHandle1, row1, column1, row2, column2)

        * Assemble and return response
        count_obj (SelectedRegions, nPins)
        Response := '(' + Message_idx + ', ' + nPins + ')'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
```

```
        break

      case 5:
        * Read all barcodes in current image
        ReadBarcodes (Image, totalBarcodeCount, Barcodes)

        Response := '(' + Message_idx + ', ' + totalBarcodeCount + ')'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')

        send_data (Socket, 'z', Barcodes, [])
        disp_message (WindowHandle1, 'Barcodes: ' + Barcodes, 'window', 40, 12, 'green',
'false')
        * set_system ('flush_graphic', 'true')
        break

      case 10:
        * Start Intrinsic Calibration
        * Assuming pixel size of 1.4u as in Basler acA4600-10
        gen_caltab (7, 7, .004, 0.5, 'iac_caltab.descr', 'iac_caltab.ps')
        gen_cam_par_area_scan_division (0.008, 0, 0.0000014, 0.0000014, image_width/2,
image_height/2, image_width, image_height, StartCamPar)
        create_calib_data ('calibration_object', 1, 1, CalibDataID)
        set_calib_data_cam_param (CalibDataID, 0, [], StartCamPar)
        set_calib_data_calib_object (CalibDataID, 0, 'iac_caltab.descr')
        I := 1
        Response := '(' + Message_idx + ', 10)'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break

      case 11:
        * Add Intrinsic calibration image
        Response_code := 11
        try
          find_calib_object (Image, CalibDataID, 0, 0, I, [], [])
          get_calib_data_observ_contours (Caltab, CalibDataID, 'caltab', 0, 0, I)
          dev_set_color ('green')
          dev_display (Caltab)
          I := I + 1
        catch (Exception)
          Response_code := 1100
        endtry
        Response := '(' + Message_idx + ',' + Response_code + ')'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break

      case 12:
        * Complete Intrinsic calibration
        try
          calibrate_cameras (CalibDataID, Error)
          get_calib_data (CalibDataID, 'camera', 0, 'params', CamParam)
          * Write the internal camera parameters to a file
          write_cam_par (CamParam, 'iac_intrinsic_camera_parameters.dat')
```

```
          Message := 'Interior camera parameters have'
          Message[1] := 'been written to file'
          disp_message (WindowHandle1, Message, 'window', 54, 12, 'green', 'false')
          clear_calib_data (CalibDataID)
          Response := '(' + Message_idx + ', 12)'
        catch (Exception)
          Response := '(' + Message_idx + ', 9912)'
        endtry
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break

      case 15:
        * Shoot target and determine extrinsic calibration
        try
          read_cam_par ('iac_intrinsic_camera_parameters.dat', CamParam)
        catch (Exception)
          Response := '(' + Message_idx + ', 9915)'
          send_data (Socket, 'z', Response, [])
          disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'red',
'false')
          break
        endtry
        *
        * Determine the external camera parameters and world coodinates from image points
        *
        * The external camera parameters can be determined from an image, where the
        * calibration plate is positioned directly on the measurement plane
        CaltabName := 'iac_caltab.descr'
        create_calib_data ('calibration_object', 1, 1, CalibDataID)
        * Here, the final camera parameters are already known and can be used instead
        * of the starting values used in the program 'camera_calibration_internal.hdev'
        set_calib_data_cam_param (CalibDataID, 0, [], CamParam)
        set_calib_data_calib_object (CalibDataID, 0, CaltabName)
        find_calib_object (Image, CalibDataID, 0, 0, 1, [], [])
        get_calib_data_observ_contours (Caltab, CalibDataID, 'caltab', 0, 0, 1)
        get_calib_data_observ_points (CalibDataID, 0, 0, 1, RCoord, CCoord, Index,
PoseForCalibrationPlate)
        dev_set_color ('green')
        dev_display (Caltab)
        dev_set_color ('red')
        disp_caltab (WindowHandle1, CaltabName, CamParam, PoseForCalibrationPlate, 1)
        dev_set_line_width (3)
        disp_circle (WindowHandle1, RCoord, CCoord, gen_tuple_const(|RCoord|,1.5))
        * To take the thickness of the calibration plate into account, the z-value
        * of the origin given by the camera pose has to be translated by the
        * thickness of the calibration plate.
        * Deactivate the following line if you do not want to add the correction.
        set_origin_pose (PoseForCalibrationPlate, 0, 0, 0.00075, PoseForCalibrationPlate)
        write_calib_data (CalibDataID, 'iac_extrinsic_cal.dat')

        Response := '(' + Message_idx + ', 15)'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break
```

```
      case 20:
        * Load model Nut Matching
        set_system ('border_shape_models', 'false')

        read_shape_model ('C:/Users/nedlecky/source/halcon/Nut Matching.shm', ModelID)
        get_shape_model_contours (ModelContours, ModelID, 1)

        smallest_rectangle1_xld (ModelContours, Row1, Column1, Row2, Column2)
        RefRow := (max(Row2)-min(Row1))/2
        RefColumn := (max(Column2)-min(Column1))/2
        vector_angle_to_rigid (0, 0, 0, RefRow, RefColumn, 0, HomMat2D)
        affine_trans_contour_xld (ModelContours, TransContours, HomMat2D)

        dev_set_color ('green')
        dev_set_draw ('margin')
        dev_display (TransContours)

        Response := '(' + Message_idx + ', 20)'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
        break

      case 21:
        * Matching Find currently loaded model
        disp_caltab (WindowHandle1, CaltabName, CamParam, PoseForCalibrationPlate, 1)
        dev_set_line_width (3)
        disp_circle (WindowHandle1, RCoord, CCoord, gen_tuple_const(|RCoord|,1.5))

        find_shape_model (Image, ModelID, rad(0), rad(360), 0.5, 1, 0.5, 'least_squares',
[5,1], 0.8, Row, Column, Angle, Score)

        * Transform the model contours into the detected positions
        FoundRow := 0
        FoundColumn := 0
        FoundAngle := 0
        FoundX := 999
        FoundY := 0
        if (|Score|==1)
          FoundRow := Row[0]
          FoundColumn := Column[0]
          FoundAngle := Angle[0]
          hom_mat2d_identity (HomMat2D)
          hom_mat2d_rotate (HomMat2D, FoundAngle, 0, 0, HomMat2D)
          hom_mat2d_translate (HomMat2D, FoundRow, FoundColumn, HomMat2D)
          affine_trans_contour_xld (ModelContours, TransContours, HomMat2D)
          dev_set_color ('green')
          dev_display (TransContours)
          * Convert camera pixel coordinates into real-world extrinsic coordinates
          image_points_to_world_plane (CamParam, PoseForCalibrationPlate, FoundRow,
FoundColumn, 1, FoundX, FoundY)
        endif

        Response := '(' + Message_idx + ', ' + FoundX +', ' + FoundY + ', ' + FoundAngle +
')'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Response: ' + Response, 'window', 26, 12, 'green',
'false')
```

```
        break

      default:
        Response := '(' + Message_idx + ', ' + (command_num_params[0]+9900) + ')'
        send_data (Socket, 'z', Response, [])
        disp_message (WindowHandle1, 'Bad Command Response: ' + Response, 'window', 26,
12, 'green', 'false')
        break

      endswitch

    catch (Command)
      OpenStatus[0] := 0
    endtry
  endwhile

  * Close up socket and grabber
  close_socket (Socket)
  close_socket (AcceptingSocket)
  close_framegrabber (AcqHandle)
endwhile
dev_close_window ()
```

Used procedures

CountPins (Image **:** SelectedRegions **: :** row1, column1, row2, column2 **)**

Short description:
Chapters:
Local procedure

```
* CountPins
* Uses tuned connection (blob) algoprithm

* Make an ROI which is a centered 80% of the image
get_image_size (Image, img_width, img_height)
roi_width := floor(img_width * 0.8)
roi_height := floor(img_height * 0.8)
row1 := floor((img_height - roi_height)/2)
column1 := floor((img_width - roi_width) / 2)
row2 := row1 + roi_height-1
column2 := column1 + roi_width-1
gen_rectangle1 (Rectangle, row1, column1, row2, column2)

* Show the ROI
```

```
dev_set_draw ('margin')
dev_display (Rectangle)

* Now threshold and connect in the ROI
reduce_domain (Image, Rectangle, ImageReduced)
threshold (ImageReduced, Bright, 80, 255)
connection (Bright, ConnectedRegions)

* Select only those blobs that qualify as pins: 120<area<1000, 5,5<width,height<40,20
select_shape (ConnectedRegions, SelectedRegions, ['area', 'width', 'height'], 'and', [120,
5, 5], [1000, 40, 20])
dev_set_draw ('fill')
dev_set_color ('green')
dev_display (SelectedRegions)

return ()
```

ReadBarcodes (Image **: : :** totalBarcodeCount, Barcodes **)**

Short description:
Chapters:
Local procedure

```
* ReadBarcodes

* Setup for barcode finding (could just do this once...)
create_bar_code_model ([], [], BarCodeHandle)
create_data_code_2d_model ('Data Matrix ECC 200', [], [], DMCodeHandle)
create_data_code_2d_model ('QR Code', [], [], QRCodeHandle)

* Find all three types of barcodes
dev_set_draw ('margin')
find_bar_code (Image, SymbolRegions128, BarCodeHandle, 'Code 128', Code128DecodedStrings)
find_data_code_2d (Image, SymbolRegionsDm, DMCodeHandle, [], [], ResultHandles,
DmDecodedStrings)
find_data_code_2d (Image, SymbolRegionsQr, QRCodeHandle, [], [], ResultHandles,
QrDecodedStrings)

* Put boxes around all the barcodes found
dev_display (SymbolRegions128)
dev_display (SymbolRegionsDm)
dev_display (SymbolRegionsQr)

* Calculate total number of barcodes found
tuple_length (Code128DecodedStrings, n1dBarcodes)
tuple_length (DmDecodedStrings, nDmCodes)
tuple_length (QrDecodedStrings, nQrCodes)
totalBarcodeCount := n1dBarcodes+nDmCodes+nQrCodes

* Concatenate all into pipe-delimited string
if (totalBarcodeCount<1)
  Barcodes := 'NOREAD'
else
  Barcodes := ''
```

```
  for Index := 0 to n1dBarcodes-1 by 1
    Barcodes := Barcodes + Code128DecodedStrings[Index] + '|'
  endfor
  for Index := 0 to nDmCodes-1 by 1
    Barcodes := Barcodes + DmDecodedStrings[Index] + '|'
  endfor
  for Index := 0 to nQrCodes-1 by 1
    Barcodes := Barcodes + QrDecodedStrings[Index] + '|'
  endfor
  * Get rid of ugly trailing '|'
  Barcodes := Barcodes{0:strlen(Barcodes)-2}
endif
return ()
```