Universitetet i Stavanger

APPLIED ROBOT TECHNOLOGY

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

# Image Aquisition: Assignment 3

***Students :***
Nourane BOUZAD
Alvaro ESTEBAN MUNOZ

***Teacher :***
Karl SKRETTING

November 11, 2021

## Contents

## 1 Introduction

In this assignment the task will be to capture an image, or a sequence of images (video), and to do some simple image processing on the image in Python. We will also continue to develop the simple image viewer and image processing framework using Python and Qt.

## 2 Use Eye camera and Python

### 2.1 Main change of the code

Since the second assignment, we manage to print all the main information on the interface. So now, it is possible to see the camera info, the errors and the number of circle in the image. Also, we develop a new option in the camera settings where it is possible to change some parameters of the image such as resolution, display mode, pixel clock and frame rate.

Here is a picture of the window created on the interface:

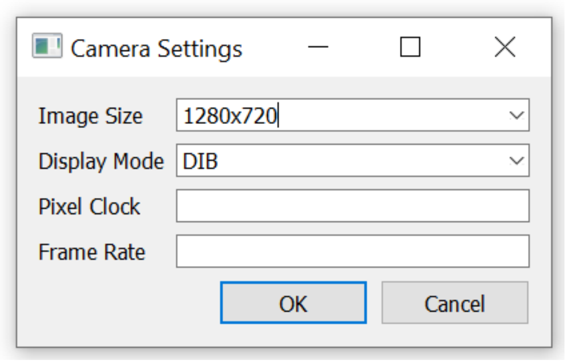

Image Size 1280x720
Display Mode DIB
Pixel Clock
Frame Rate

However, we haven't managed to make all the options of this menu to work, it is only possible to change the Display Mode and the Frame Rate. We will consider fixing it on the next assignment, since our time budget for this one is completed.

This options dialog is a new class implemented by us based on QWidget class from python QT.

```python
import pyueye_example_camera

from PyQt5.QtWidgets import (QWidget, QDialog, QInputDialog,
    QFormLayout, QLineEdit, QDialogButtonBox, QComboBox)

class QInputDialogJ(QWidget):
    """Display a dialog that asks for few parameters
    to thath will be returned
    """
    def __init__(self, mainApp):
        super().__init__()
        super().setWindowTitle("Camera Settings")
        super().resize(400, 200)
        # To call the mainApp methods
        self.mainApp = mainApp

        self.imageSize = QComboBox(self)
        self.imageSize.addItems(["640x480", "800x480", "1280x720"])
        self.imageSize.setEditable(True)
        self.imageSize.setCurrentText("1280x720")

        self.displayMode = QComboBox(self)
        self.displayMode.addItems(["DIB", "Direct3D", "OpenGL"])
        self.displayMode.setEditable(True)
        self.displayMode.setCurrentText("DIB")

        self.pixelClock = QLineEdit(self)
        self.frameRate = QLineEdit(self)
        self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok |
           QDialogButtonBox.Cancel, self)

        self.layout = QFormLayout(self)
        self.layout.addRow("Image Size", self.imageSize)
        self.layout.addRow("Display Mode", self.displayMode)
        self.layout.addRow("Pixel Clock", self.pixelClock)
        self.layout.addRow("Frame Rate", self.frameRate)
        self.layout.addWidget(self.buttonBox)

        self.buttonBox.accepted.connect(self.accept)
        self.buttonBox.rejected.connect(self.reject)

        return

    def accept(self):
        """Close the dialog and tell the mainApp to change options
        """
        self.close()
        returnedValue = (
            self.imageSize.currentText(),
            self.displayMode.currentIndex(),
            self.pixelClock.text(),
            self.frameRate.text()
        )

        self.mainApp.changeOptions(returnedValue)

        return

    def reject(self):
        """Close the dialog and return None
        """
        self.close()
        return
```

We also implemented a new menu for recording options where we can display frames that are being recorded by the image and another option to find dice in real time. To achieve this task, we used a script proposed by the official IDS webpage and changed it a bit, this script can be downloaded here. We think it needs more changes to be perfectly adapted to our application, nevertheless, we will take care of it on the next assignment.

## 2.2 Code of the new version

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
#
# ../ELE610/py3/appSimpleImageViewer.py
#
#  Simple program that uses Qt to display an image, only basic
   options.
#    File menu: Open File, and Quit
#    Scale menu: Scale Up, and Scale down
#  No status line at bottom.
#
# Karl Skretting, UiS, September - November 2018, November 2020

# Example on how to use file:
# (C:\...\Anaconda3) C:\..\py3> activate py38
# (py38) C:\..\py3> python appSimpleImageViewer.py

_appFileName = "appSimpleImageViewerj"
_author = " lvaro  Esteban Mu oz & Nourane Bouzad"
_version = "2021.00.01"


# Useful libraries
import sys
import numpy as np
import cv2
import time
import threading

# QT libraries
import qimage2ndarray
from PyQt5.QtCore import (QT_VERSION_STR, QRect, Qt)
from PyQt5.QtGui import QPixmap, QTransform, QImage
from PyQt5.QtWidgets import (QApplication, QWidget, QMainWindow,
          QGraphicsScene, QGraphicsView, QGraphicsPixmapItem,
             QAction, QFileDialog, QInputDialog, QMessageBox,
             QErrorMessage)

# Pyueye libraries
```

```python
from pyueye import ueye
from pyueye_example_utils import ImageData, ImageBuffer
from pyueye_example_camera import Camera
from SimpleLive_Pyueye_OpenCV import record_video

# My libraries
from QInputDialogJ import QInputDialogJ

# Path to the images folder
myPath = './images'

class MainWindow(QMainWindow):
    """MainWindow class for a simple image viewer."""
    def __init__(self, parent = None):
        """Initialize the main window object with title, location
            and size,
        an empty image (pixmap), empty scene and empty view
        """
        super().__init__(parent)
        self.setWindowTitle('Simple Image Viewer')
        self.setGeometry(150, 50, 1400, 800)  # initial window
            position and size

        # Camera
        self.cam = None
        self.camOn = False
        self.cInfo = None
        self.frameRate = 30

        # Scene
        self.curItem = None
        self.pixmap = QPixmap()  # a null pixmap
        self.scene = QGraphicsScene()
        self.view = QGraphicsView(self.scene, parent=self)
        self.view.setGeometry(0, 20, self.width(), self.height()
            -20)
        self.initMenu()

        self.msgBox = QMessageBox()   # Dialog to print messages
        self.msgError = QErrorMessage()    # Dialog to print
            error messages

        # Initialization of the camera options dialog
        self.cameraOps = QInputDialogJ(self)
        self.dispModes = [ueye.IS_SET_DM_DIB, ueye.
            IS_SET_DM_DIRECT3D, ueye.IS_SET_DM_OPENGL]

        return

    def initMenu(self):
        """Set up the menu for main window: File with Open and
           Quit, Scale with + and -."""

        # SUBMENUS CONFIGURATION #
        # FILE MENU
        qaOpenFile = QAction('Open File', self)
        qaOpenFile.setShortcut('Ctrl+O')
        qaOpenFile.setStatusTip('Open (image) File using dialog
           box')
        qaOpenFile.triggered.connect(self.openFile)

        qaSaveFile = QAction('Save File', self)
        qaSaveFile.setShortcut('Ctrl+S')
        qaSaveFile.setStatusTip('Save image to a file')
        qaSaveFile.triggered.connect(self.saveFile)

        qaCloseWin = QAction('Close Window', self)
        qaCloseWin.setShortcut('Ctrl+Q')
        qaCloseWin.setStatusTip('Close and quit program')
        qaCloseWin.triggered.connect(self.closeWin)

        # CAMERA MENU
        qaCameraOn = QAction('Camera On', self)
        qaCameraOn.triggered.connect(self.cameraOn)

        qaCameraOff = QAction('Camera Off', self)
        qaCameraOff.triggered.connect(self.cameraOff)

        qaGetShot = QAction('Take a Snapshot', self)
        qaSaveFile.setShortcut('Ctrl+P')
        qaGetShot.triggered.connect(self.getShot)

        qaCameraInfo = QAction('Print Camera Info', self)
        qaCameraInfo.triggered.connect(self.cameraInfo)

        qaCameraOptions = QAction('Change Camera Options', self)
        qaCameraOptions.triggered.connect(self.cameraOps)

        #qaRecordVideo = QAction('Record video', self)
        #qaRecordVideo.triggered.connect(self.captureVideo)

        #qaStopVideo = QAction('Stop video', self)
        #qaStopVideo.triggered.connect(self.stopVideo)

        # RECORD MENU
        qaRecordVideo = QAction('Record Video', self)
        qaRecordVideo.triggered.connect(self.captureVideo)

        qaFindDices_Video = QAction('Find dices in real time',
           self)
        qaFindDices_Video.triggered.connect(self.findDices_Video)

        # SCALE MENU
        qaScaleUp = QAction('Scale Up', self)
        qaScaleUp.setShortcut('Ctrl++')
        qaScaleUp.triggered.connect(self.scaleUp)

        qaScaleDown = QAction('Scale Down', self)
        qaScaleDown.setShortcut('Ctrl+-')
        qaScaleDown.triggered.connect(self.scaleDown)

        # EDIT MENU
        qaCropImg = QAction('Crop Image', self)
        qaCropImg.setShortcut('Ctrl+R')
        qaCropImg.setStatusTip('Crop image to a selected area')
        qaCropImg.triggered.connect(self.cropImg)

        qaGrayScale = QAction('Gray Scale', self)
        qaGrayScale.setShortcut('Ctrl+G')
        qaGrayScale.setStatusTip('Turn the image into a gray
           scale image')
        qaGrayScale.triggered.connect(self.grayScale)

        qaBlackDots = QAction('Binary image', self)
        qaBlackDots.setShortcut('Ctrl+B')
        qaBlackDots.setStatusTip('Turn the image into binary
           color')
        qaBlackDots.triggered.connect(self.blackDots)

        # DICE MENU
        qaHoughCircles = QAction('Find Circles', self)
        qaHoughCircles.setShortcut('Ctrl+C')
        qaHoughCircles.setStatusTip('Find circles on the image')
        qaHoughCircles.triggered.connect(self.houghcircles)

        qaFindDices = QAction('Find Dices', self)
        qaFindDices.setStatusTip('Find circles in each dice on
           the image')
        qaFindDices.triggered.connect(self.findDices)

        # MENU BAR CONFIGURATION
        mainMenu = self.menuBar()

        fileMenu = mainMenu.addMenu('&File')
        fileMenu.addAction(qaOpenFile)
        fileMenu.addAction(qaCloseWin)
        fileMenu.addAction(qaSaveFile)

        cameraMenu = mainMenu.addMenu('&Camera')
        cameraMenu.addAction(qaCameraOn)
```

```python
        cameraMenu.addAction(qaCameraOff)
        cameraMenu.addAction(qaGetShot)
        cameraMenu.addAction(qaCameraInfo)
        cameraMenu.addAction(qaCameraOptions)
        #cameraMenu.addAction(qaRecordVideo)
        #cameraMenu.addAction(qaStopVideo)

        videoMenu = mainMenu.addMenu('&Video')
        videoMenu.addAction(qaRecordVideo)
        videoMenu.addAction(qaFindDices_Video)

        scaleMenu = mainMenu.addMenu('&Scale')
        scaleMenu.addAction(qaScaleUp)
        scaleMenu.addAction(qaScaleDown)

        editMenu = mainMenu.addMenu('&Edit')
        editMenu.addAction(qaCropImg)
        editMenu.addAction(qaGrayScale)
        editMenu.addAction(qaBlackDots)

        diceMenu = mainMenu.addMenu('&Dice')
        diceMenu.addAction(qaHoughCircles)
        diceMenu.addAction(qaFindDices)

        return

# Methods for File menu
    def openFile(self):
        """Use the Qt file open dialog to select an image to open as a pixmap,
        The pixmap is added as an item to the graphics scene which is shown in the graphics view.
        The view is scaled to unity.
        """
        options = QFileDialog.Options()
        options |= QFileDialog.DontUseNativeDialog    # make dialog appear the same on all systems
        flt = "All jpg files (*.jpg);;All bmp files (*.bmp);;All png files (*.png);;All files (*)"
        (fName, used_filter) = QFileDialog.getOpenFileName(parent=self, caption="Open image file",
            directory=myPath, filter=flt, options=options)
        #
        if (fName != ""):
            if self.curItem:
                self.scene.removeItem(self.curItem)
                self.curItem = None
            #end if
            self.pixmap.load(fName)
            # If the file does not exist or is of an unknown
              format, the pixmap becomes a null pixmap.
            if self.pixmap.isNull():
                self.setWindowTitle('Image Viewer (error for
                  file %s)' % fName)
                self.view.setGeometry( 0, 20, self.width(),
                  self.height()-20 )
            else: # ok
                self.curItem = QGraphicsPixmapItem(self.pixmap)
                self.scene.addItem(self.curItem)
                self.setWindowTitle('Image Viewer: ' + fName)
                self.view.setTransform(QTransform())  #
                  identity (for scale)
            #end if
        #end if
        return

    def closeWin(self):
        """Quit program."""
        self.msgBox.setText("Close the main window and quit
          program.")
        self.msgBox.exec()
        self.close()
        return

    def saveFile(self):
        options = QFileDialog.Options()

        flt = "All jpg files (*.jpg);;All bmp files (*.bmp);;All
          png files (*.png);;All files (*)"
        (fName, used_filter) = QFileDialog.getSaveFileName(self,
          caption="Save image file as",
            directory=myPath, filter=flt, options=options)

        if (fName != ""):
            if self.pixmap.save(fName):
                self.msgBox.setText(f"Saved image into file {
                  fName}")
                self.msgBox.exec()
            else:
                self.msgError.showMessage("Failed to save the
                  image")

        return

# Methods for Camera menu
    def cameraOn(self):
        """Turn IDS camera on."""
        if not self.camOn:
```

```python
            # Initialize the camera
            self.cam = Camera()
            self.cam.init()
            self.cInfo = ueye.CAMINFO()
            nRet = ueye.is_GetCameraInfo(self.cam.handle(), self.cInfo)

            # Set the color mode for the camera
            self.cam.set_colormode(ueye.IS_CM_BGR8_PACKED)

            # This function is currently not supported by the camera models USB 3 uEye XC and XS.
            self.cam.set_aoi(0, 0, 720, 1280)  # but this is the size used
            self.cam.alloc(3)  # argument is number of buffers
            self.camOn = True

            # Print message
            self.msgBox.setText('Camera started.')
            self.msgBox.exec()
        else:
            self.msgError.showMessage("Camera is already on")

        return

    def copy_image(self, image_data):
        """Copy an image from camera memory to numpy image array."""

        # Variable to store the image (numpy array representation)
        npImage = None

        tempBilde = image_data.as_1d_image()
        if np.min(tempBilde) != np.max(tempBilde):
            npImage = np.copy(tempBilde[:,:,[2,1,0]])  # or [2,1,0] ??  RGB or BGR?
        else:
            npImage = np.array([])  # size == 0

        image_data.unlock()  # Free memory

        return npImage

    def cameraInfo(self):
        """Print information of the camera"""
        if self.camOn:

            infoStr = "CAMERA INFORMATION:\n"
            infoStr += ("   Camera serial no.:     %s\n" % self.
               cInfo.SerNo.decode('utf-8')) # 12 byte
            infoStr += ("   Camera ID:             %s\n" % self.
               cInfo.ID.decode('utf-8')) # 20 byte
            infoStr += ("   Camera Version:        %s\n" % self.
               cInfo.Version.decode('utf-8')) # 10 byte
            infoStr += ("   Camera Date:           %s\n" % self.
               cInfo.Date.decode('utf-8')) # 12 byte
            infoStr += ("   Camera Select byte:    %i\n" % self.
               cInfo.Select.value) # 1 byte
            infoStr += ("   Camera Type byte:      %i\n" % self.
               cInfo.Type.value) # 1 byte
            infoStr += "\n"

            d = ueye.double()
            retVal = ueye.is_SetFrameRate(self.cam.handle(),
               self.frameRate, d)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Frame rate set to
                                        %8.3f fps' % d)
                infoStr += '\n'
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
               IS_EXPOSURE_CMD_GET_EXPOSURE_DEFAULT, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Default setting for the exposure
                    time  %8.3f ms' % d)
                infoStr += '\n'
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
               IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MIN, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Minimum exposure time
                                    %8.3f ms' % d)
                infoStr += '\n'
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
               IS_EXPOSURE_CMD_GET_EXPOSURE_RANGE_MAX, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Maximum exposure time
                                    %8.3f ms' % d)
                infoStr += '\n'
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
               IS_EXPOSURE_CMD_GET_EXPOSURE, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Currently set exposure time
                              %8.3f ms' % d)
                infoStr += '\n'
            d =  ueye.double(25.0)
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
               IS_EXPOSURE_CMD_SET_EXPOSURE, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Tried to changed exposure time
                  to      %8.3f ms' % d)
                infoStr += '\n'
            retVal = ueye.is_Exposure(self.cam.handle(), ueye.
              IS_EXPOSURE_CMD_GET_EXPOSURE, d, 8)
            if retVal == ueye.IS_SUCCESS:
                infoStr += ('  Currently set exposure time
                          %8.3f ms' % d)
                infoStr += '\n'

            self.msgBox.setText(infoStr)
            self.msgBox.exec()
        else:
            self.msgBox.setText('Camera is not on, please turn
              it on using Camera -> Camera On.')
            self.msgBox.exec()

        return

    def cameraOff(self):
        """Turn IDS camera off"""
        if self.camOn:

            self.cam.exit()
            self.camOn = False

            self.msgBox.setText('Camera stopped')
            self.msgBox.exec()

        return

    def getShot(self):

        if self.camOn:
            imBuffer = ImageBuffer()

            # TODO self.cam.alloc()
            self.cam.freeze_video(True)        # Freeze the
              video (Save the frame on memory)
            retVal = ueye.is_WaitForNextImage(self.cam.handle(),
               1000, imBuffer.mem_ptr, imBuffer.mem_id)

            if retVal == ueye.IS_SUCCESS:
                # Copy the image to a numpy array
                npImage = self.copy_image(ImageData(self.cam.
                  handle(), imBuffer))

                # Set all the items for the scene
                image = qimage2ndarray.array2qimage(npImage)
                self.pixmap = QPixmap.fromImage(image)
                self.scene.removeItem(self.curItem)
                self.curItem = QGraphicsPixmapItem(self.pixmap)
                self.scene.addItem(self.curItem)
            else:
                self.msgError.showMessage('There was an error
                   getting the image')

        else:
            self.msgError.showMessage("Camera is not connected.
               Please remember to turn on camera using Camera
               --> Camera On.")

        return

    def cameraOps(self):
        """Display the camera options dialog"""
        if self.camOn:
            self.cameraOps.show()
        else:
            self.msgError.showMessage("Camera is not connected.
               Please remember to turn on camera using Camera
               --> Camera On.")

        return

    def changeOptions(self, options):
        """Change the camera options"""
        # Get the input from the user
        imSize, dispMode, pixClock, frameRate = options

        # Set the types
        rate = ueye.DOUBLE(int(frameRate))
        self.frameRate = ueye.DOUBLE()

        # TODO Update camera options
        #self.cam.set_aoi(0, 0, int(imSize.split('x')[0]), int(
           imSize.split('x')[1]))

        # Set the display mode
        ueye.is_SetDisplayMode(self.cam.handle(), self.dispModes[
           dispMode])

        # TODO Check that the value is an integer
        # Set pixel clock
        #ueye.is_PixelClock(self.cam.handle(), ueye.
           IS_PIXELCLOCK_CMD_SET, int(pixClock), ueye.sizeof(ueye
           .INT))
        # Set the frame rate
        ueye.is_SetFrameRate(self.cam.handle(), rate, self.
           frameRate)
    def captureVideo(self):

        if self.camOn:
            self.cameraOff()

        record_video(process=False)

        return

    def stopVideo(self):
        if self.camOn:
            # TODO Stop video
            pass
        else:
            self.msgError.showMessage("Camera is not connected.
               Please remember to turn on camera using Camera
               --> Camera On.")

    def findDices_Video(self):

        if self.camOn:
            self.cameraOff()

        record_video(process=True)

        return


# Methods for Scale menu
    def scaleUp(self):
        """Scale up the view by factor 2"""
        if not self.pixmap.isNull():
            self.view.scale(2,2)
        return

    def scaleDown(self):
        """Scale down the view by factor 0.5"""
        if not self.pixmap.isNull():
            self.view.scale(0.5,0.5)
        return

# Methods for Edit menu
    def cropImg(self):

        # TODO Perfectionate the input dialogs
        # Ask for the parameters to crop the image
        x = QInputDialog.getInt(self, 'Crop area', 'Introduce x
           coordinate for the left top corner of the cropped area
           ')
        y = QInputDialog.getInt(self, 'Crop area', 'Introduce y coordinate for the left top corner of the cropped area')
        height = QInputDialog.getInt(self, 'Crop area', 'Introduce height for the area to crop')
        width = QInputDialog.getInt(self, 'Crop area', 'Introduce widht for the area to crop')

        # Create the new form for the image
        rect = QRect(x[0], y[0], width[0], height[0])
        if not self.pixmap.isNull():
            # Clear the scene
            self.scene.removeItem(self.curItem)
            self.curItem = None

            # Copy the original image in the cropping rectangle
            cropped = self.pixmap.copy(rect)

            # Set the cropped image as the actual image
            self.pixmap = cropped
            self.curItem = QGraphicsPixmapItem(cropped)
            self.scene.addItem(self.curItem)

        return

    def grayScale(self):

        if not self.pixmap.isNull():
            image = self.pixmap.toImage()

            npImage = qimage2ndarray.rgb_view(image)
            npImage = cv2.cvtColor(npImage, cv2.COLOR_RGB2GRAY)
            image = qimage2ndarray.array2qimage(npImage)

            self.pixmap = QPixmap.fromImage(image)
            self.scene.removeItem(self.curItem)
            self.curItem = QGraphicsPixmapItem(self.pixmap)
            self.scene.addItem(self.curItem)

        return

    # Methods for Dice menu
    def blackDots(self):
        image = self.pixmap.toImage()
        npImage = qimage2ndarray.rgb_view(image)
        npImage = cv2.cvtColor(npImage, cv2.COLOR_BGR2GRAY)
        thresh = 65
        im_bin = cv2.threshold(npImage, thresh, 255, cv2.THRESH_BINARY)[1]
        image = qimage2ndarray.array2qimage(im_bin)

        # Set all the items (Maybe we should create a method for
           that)
        self.pixmap = QPixmap.fromImage(image)
        self.scene.removeItem(self.curItem)
        self.curItem = QGraphicsPixmapItem(self.pixmap)
        self.scene.addItem(self.curItem)

    def houghcircles(self):

        image = self.pixmap.toImage()
        npImage = qimage2ndarray.rgb_view(image)
        npImage = cv2.cvtColor(npImage, cv2.COLOR_BGR2GRAY)
        #npImage = cv2.medianBlur(npImage,5)

        circles = cv2.HoughCircles(npImage, cv2.HOUGH_GRADIENT,
           dp=1.2, minDist=20,
             param1=60, param2=40, minRadius=10, maxRadius=50)
        npImage = cv2.cvtColor(npImage, cv2.COLOR_GRAY2BGR)

        list_number_circles=[]
        if circles is not None:
             circles = np.uint16(np.around(circles))
             for i in circles[0,:]:
                  cv2.circle(npImage,(i[0],i[1]),i[2],(0,255,0)
                     ,2)
                  cv2.circle(npImage,(i[0],i[1]),2,(0,0,255),3)
                  list_number_circles.append(len(circles))

             number_circles= sum(list_number_circles)

             # Print number of circles
             self.msgBox.setText("Number of circles: "+str(
                number_circles))
             self.msgBox.exec()
        else:
             self.msgBox.setText("No circles found")
             self.msgBox.exec()


        image = qimage2ndarray.array2qimage(npImage)

        # Set all the items for the scene
        self.pixmap = QPixmap.fromImage(image)
        self.scene.removeItem(self.curItem)
        self.curItem = QGraphicsPixmapItem(self.pixmap)
        self.scene.addItem(self.curItem)

        return
    def findDices(self):
        """Find dices in active image using ??."""
        #
        # -- your code may be written in between the comment
            lines below --
        # find dices by looking for large rectangles (squares) in
             the image matching each color
        # each color can be a small set of color point that can
            be loaded into custom color list
        # for each color (point set)
        #    find distance to this color (point set) and
            threshold
        #    perhaps morphological operations on this binary
            image, erode and dilate
        #    find large area (and check it is almost square)
        #    (to find eyes too, the number of same size black
            wholes inside the square could be found)
        #    print results, or indicate it on image
        #

        # Get the numpy array version of the image
        image = self.pixmap.toImage()
        npImage = qimage2ndarray.rgb_view(image)

        # Convert to gray scale
        npImage = cv2.cvtColor(npImage, cv2.COLOR_BGR2GRAY)

        # Sharpen image
        kernel = np.array([[-1, -1, -1], [-1, 8, -1], [-1, -1,
            -1]])/8
        sharpened_img = cv2.filter2D(npImage, -1, kernel)

        # Find edges with canny edge detector
        #edged_img = cv2.Canny(sharpened_img, 30, 200)

        # Turn the numpy image to a QImage
        image = qimage2ndarray.array2qimage(sharpened_img)

        # Set the scene
        self.pixmap = QPixmap.fromImage(image)
        self.scene.removeItem(self.curItem)
        self.curItem = QGraphicsPixmapItem(self.pixmap)
        self.scene.addItem(self.curItem)

        return


# methods for 'slots'
```

```
    def resizeEvent(self, arg1):
        """Make the size of the view follow any changes in the
           size of the main window.
        This method is a 'slot' that is called whenever the size
           of the main window changes.
        """
        self.view.setGeometry( 0, 20, self.width(), self.height()
           -20 )
        return
#end class MainWindow

if __name__ == '__main__':
    print("%s: (version %s), path for images is: %s" % (
        _appFileName, _version, myPath))
    print("%s: Using Qt %s" % (_appFileName, QT_VERSION_STR))
    mainApp = QApplication(sys.argv)
    mainWin = MainWindow()
    mainWin.show()
    sys.exit(mainApp.exec_())
```

## 2.3 Change that could be done

For this task we have been able to count the number of eyes but for every dices present in the image. The idea would be to do it for each dice. To do so, we have thought that the most robust and simplest approach would be to associate each circles to a dice regarding his color, so the intensity number of the pixel. Since each dice has different color, once we identify the different dice with their associated color, we could implement the program which will count the number of circle.

Regarding the record menu, we consider displaying the frames that are being recorded on the application's interface, (currently they are being displayed in another window). In addition, our code is not perfect, that's why we think we also need to clean it and make it more readable,

# 3 Time table

| Members | Nourane Bouzad | Alvaro Esteban Munoz |
|---|---|---|
| Time used | 10h | 10h |