APPLIED ROBOT TECHNOLOGY

ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Image Aquisition: Assignment 4

Students :
Nourane BOUZAD
Alvaro ESTEBAN MUNOZ

Teacher :
Karl SKRETTING

January 2, 2023

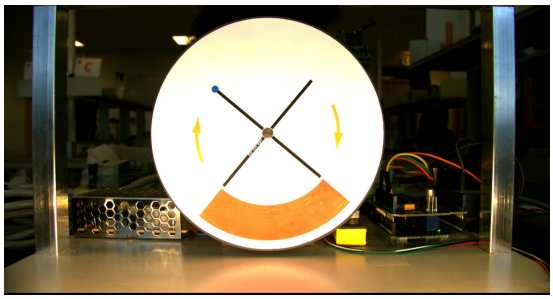
Contents

1 Introduction

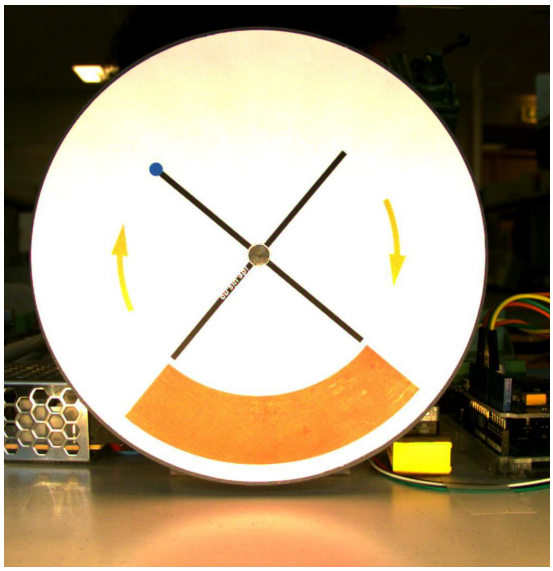
In this assignment, we will work with a new camera and a new object. The object, here, is a disk faced directly towards the camera. The disk is driven by a motor and the speed can be adjusted. Using Python and image processing (OpenCV) the task is to find and show the angular speed of the disk (rotations per minute, rpm) based on one image, or a sequence of images (video). On the new (the smaller one) camera rig, it should be possible to read out the true speed. We will also continue to develop the simple image viewer and image processing framework using Python and Qt.

2 Angle of disk in image

a) Using the IDS µEye Cockpit program to adjust the camera options, we've captured the following image of the disk:



b) Here, we've cropped the image as follows:

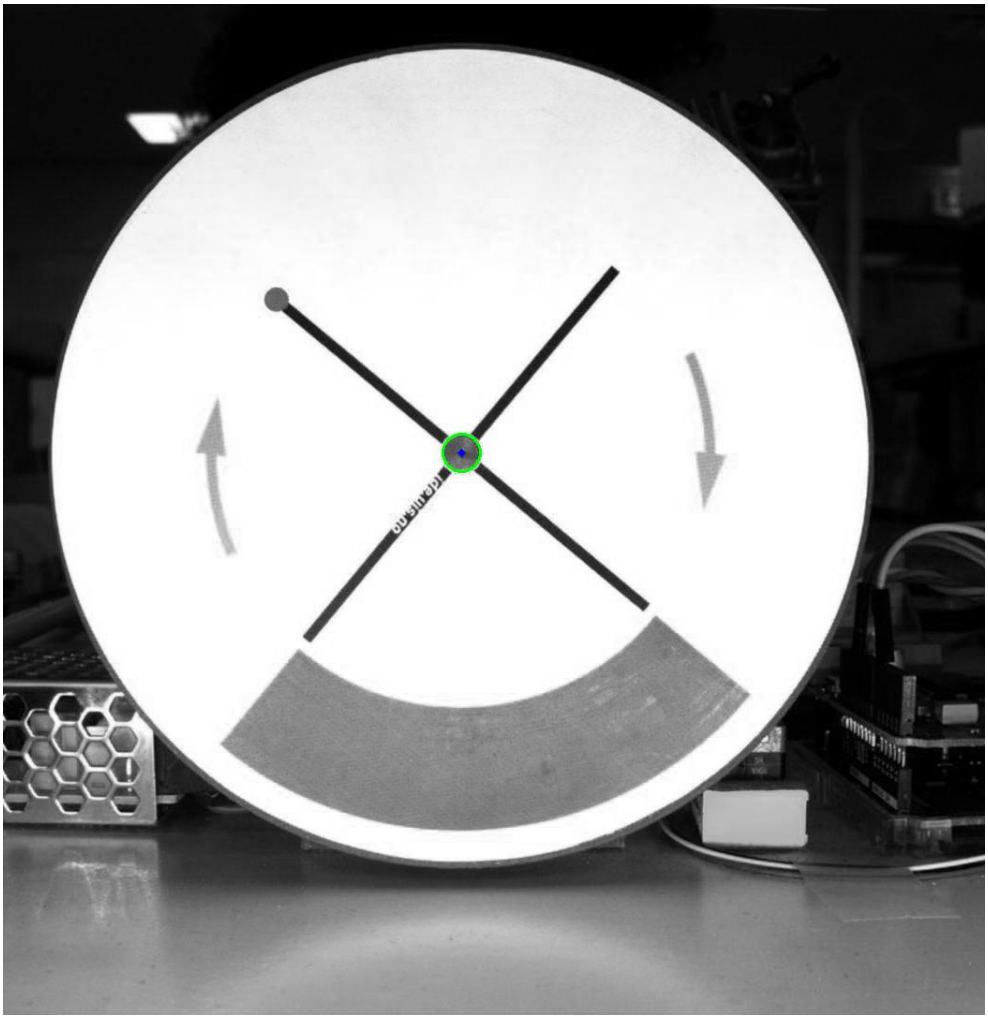


For that, we use the program implemented in the first assignment. To do so, we use the following parameters:

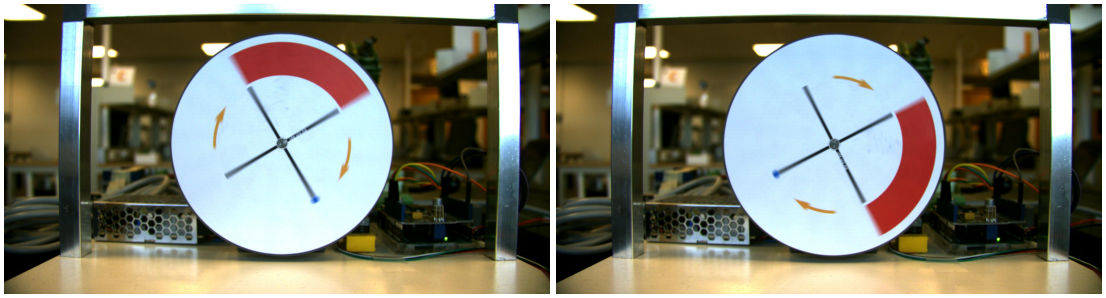
- width : 1048
- height : 1080

c) For this question, we choose to use the program setting in the third assignment. To find only the center circle, we set the following parameter:

$$dp = 1.2, minDist = 20, param1 = 60, param2 = 40, minRadius = 17, maxRadius = 25$$

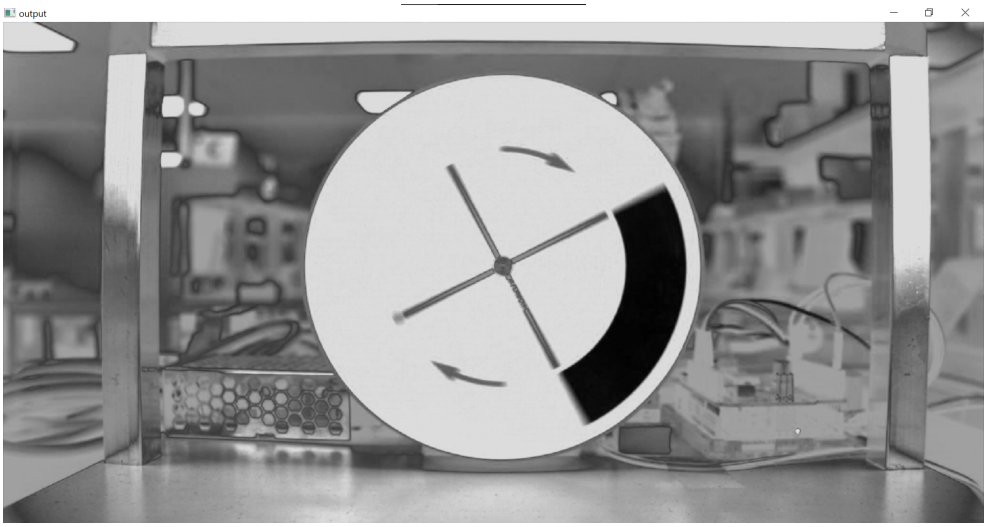


d) To figure out the angle manually, we took 2 pictures with different exposure time.



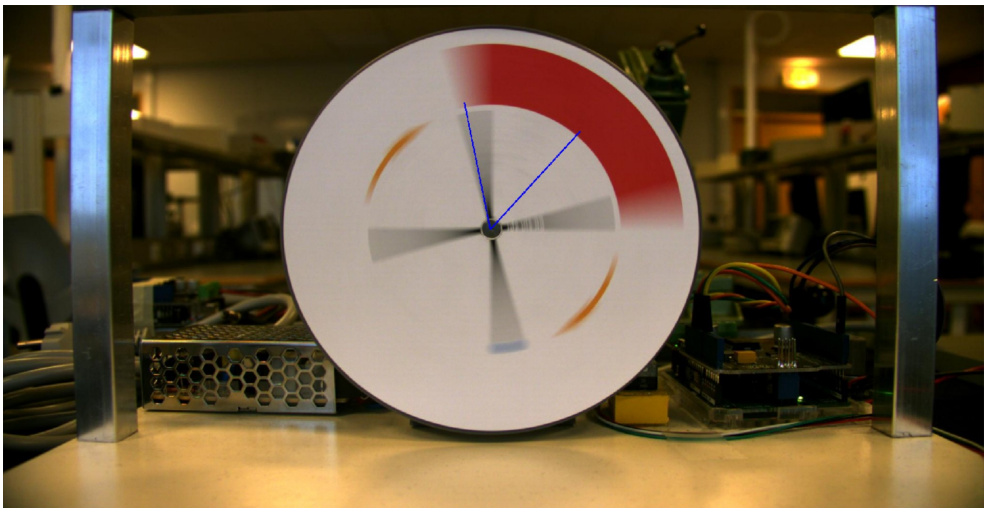
Comparing the two images, we can see that we have approximately 90 degrees (a bit more).

e) The idea here is to have the image of the disk and an image fully red, which will be subtracted to the original image. The interesting part will appear as black, detecting this part

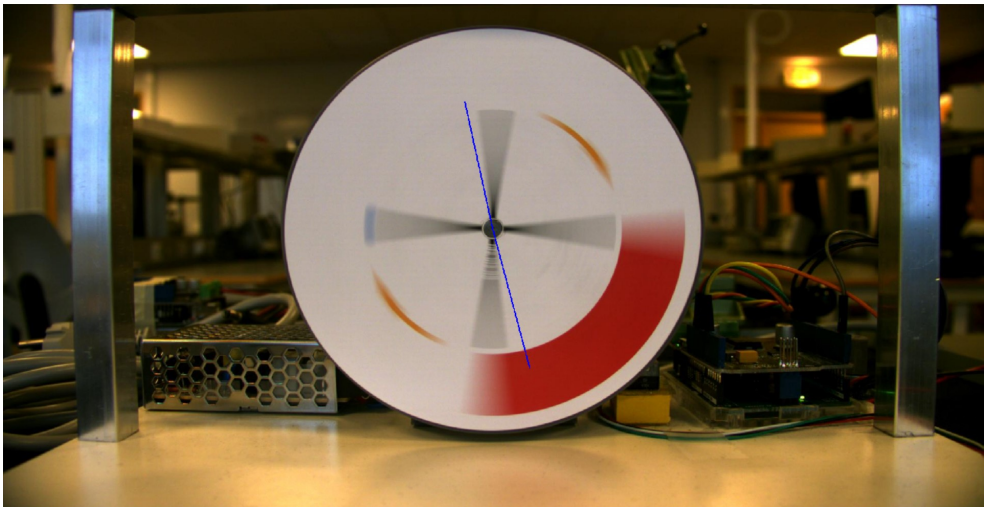


3 Camera in trigger mode

a) After 10 ms, we have an angle of 58º.



b) After 50 ms, we have an angle of 182º.



c) The angle difference between both is:

$$\theta_2 - \theta_2 = 182 - 58 = 124$$

The time difference between both is:

$$t_2 - t_1 = 50 - 10 = 40ms$$

d) To find the rotation speed, it's more accurate to use the difference between the 2 previous points as a mean value, making the result more precise. To do so, we have the angle difference and the time difference, the quotient of these values give us the rotation speed in degrees per ms, so to convert it to the wanted unit we apply the following formula:

$$\omega = \frac{d\theta}{dt} = \frac{124 * 60}{40 * 10^{-3} * 360} = 516.6rpm$$

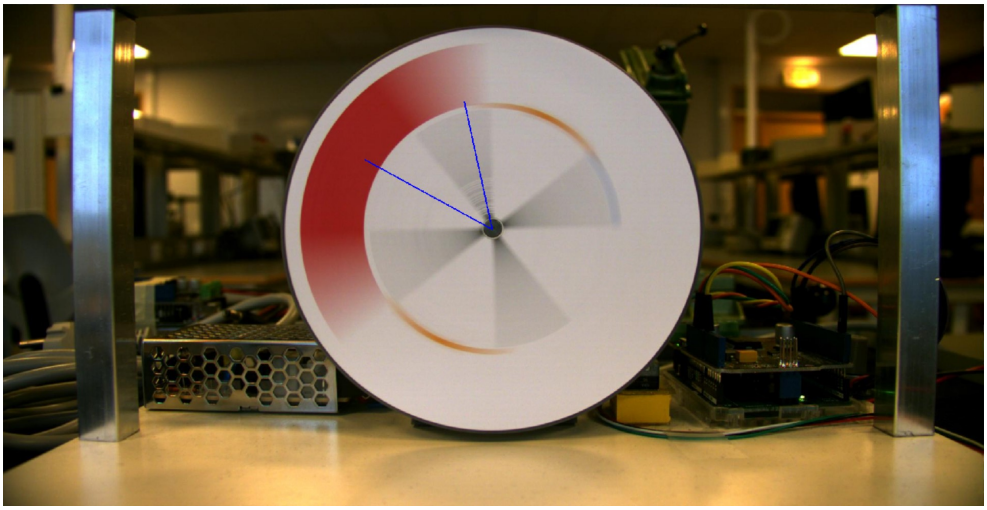
Since one revolution is 360° and 1 sec is $\frac{1}{60}$ min.

e) We expect an angle of 0° because we are taking the stopped disk as starting point, so we don't have time to detect an angle.

f) Increasing the rotation speed of the disk, we got an angle of 316°. So, to know the rotation speed, we apply the following formula:

$$\omega = \frac{316 * 60}{50.10^{-3} * 360} = 1053.3rpm$$

The result may have some errors due to the delay between the capture of the image and the processing by the code.



4 Python program

```
#!/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
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
from findAngle import findAngle

# 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
        self.triggerOn = False

        # 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)

        qaActivateTrigger = QAction('Activate Trigger', self)
        qaActivateTrigger.triggered.connect(self.activateTrigger)

        qaDeactivateTrigger = QAction('Deactivate Trigger', self)
```

```
        qaDeactivateTrigger.triggered.connect(self.
            deactivateTrigger)

        # 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)

        # ANGLE MENU
```

```
        qaAngle = QAction('Find Angle', self)
        qaAngle.setShortcut('Ctrl+A')
        qaAngle.setStatusTip('Find the angle of the dice')
        qaAngle.triggered.connect(self.find_angle)
        qDiskCenter = QAction('Find Center', self)
        qDiskCenter.setStatusTip('Find the center of the dice')
        qDiskCenter.triggered.connect(self.find_center)
        qRotationSpeed = QAction('Find Rotation Speed', self)
        qRotationSpeed.setStatusTip('Find the rotation speed of
           the disk')
        qRotationSpeed.triggered.connect(self.
           compute_rotation_speed)

        # 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)
        cameraMenu.addAction(qaCameraOff)
        cameraMenu.addAction(qaGetShot)
        cameraMenu.addAction(qaCameraInfo)
        cameraMenu.addAction(qaCameraOptions)
        cameraMenu.addAction(qaActivateTrigger)
        cameraMenu.addAction(qaDeactivateTrigger)

        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)

        angleMenu = mainMenu.addMenu('&Angle')
        angleMenu.addAction(qaAngle)
        angleMenu.addAction(qDiskCenter)
```

```
            angleMenu.addAction(qRotationSpeed)

            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:

                  # 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'
340             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)
345             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:
350                 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:
355                 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)
360                 infoStr += '\n'

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

            return

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

                self.cam.exit()
```

```
375                 self.camOn = False

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

380             return

        def getShot(self):

             if self.camOn:
385                 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)
390
                    if retVal == ueye.IS_SUCCESS:
                         # Copy the image to a numpy array
                         npImage = self.copy_image(ImageData(self.cam.
                            handle(), imBuffer))

395                      # 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)
400                      self.scene.addItem(self.curItem)
                    else:
                         self.msgError.showMessage('There was an error
                            getting the image')

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

             return

        def cameraOps(self):
410          """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.")
415
             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 activateTrigger(self):
        """Activate the trigger"""
        if self.camOn:
            ueye.is_SetExternalTrigger(self.cam.handle(), ueye.
                IS_SET_TRIGGER_HI_LO)
            ueye.is_SetTriggerDelay(self.cam.handle(), 50000)
            self.triggerOn = True
        else:
            self.msgError.showMessage("Camera is not connected.
                Please remember to turn on camera using Camera
                --> Camera On.")

        return

    def deactivateTrigger(self):
        """Deactivate the trigger"""
        if self.camOn:
            ueye.is_SetExternalTrigger(self.cam.handle(), ueye.
                IS_SET_TRIGGER_OFF)
            ueye.is_SetTriggerDelay(self.cam.handle(), 0)
            self.triggerOn = False
        else:
```

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

            return

# Methods for record menu
    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, minRadius=None, maxRadius=None):

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

            # TODO ask parameters in a dialog (430, 434)
            if minRadius is None or maxRadius is None:
                minRadius = QInputDialog.getInt(self, 'Find circles'
                    , 'Introduce minimun radius to find the circles')
                    [0]
                maxRadius = QInputDialog.getInt(self, 'Find circles'
                    , 'Introduce maximun radius to find the circles')
                    [0]

            circles = cv2.HoughCircles(npImage, cv2.HOUGH_GRADIENT,
                dp=1.2, minDist=20,
                  param1=60, param2=40, minRadius=minRadius, maxRadius
                    =maxRadius)
            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 angle menu
        def find_center(self):
            self.houghcircles(430, 434)

        def find_angle(self):

            # Read the initial image to find the starting point
            start_im = cv2.imread('.\\appImageViewer4J\images\
               Initial_angle_disk.jpg')

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

            # OpenCV works with BGR images, so we convert it to BGR
            end_im = cv2.cvtColor(end_im, cv2.COLOR_RGB2BGR)

            theta, angle_im = findAngle(start_im, end_im)

            if theta is not None:

                self.msgBox.setText("Angle: "+str(theta))
                self.msgBox.exec()

                # Turn the numpy image to a QImage
                angle_im = cv2.cvtColor(angle_im, cv2.COLOR_BGR2RGB)
                image = qimage2ndarray.array2qimage(angle_im)

                # Set the scene
                self.pixmap = QPixmap.fromImage(image)
                self.scene.removeItem(self.curItem)
                self.curItem = QGraphicsPixmapItem(self.pixmap)
                self.scene.addItem(self.curItem)
            else:
                self.msgBox.setText("No angle found")
```

```
                self.msgBox.exec()

            return theta

        def compute_rotation_speed(self):
            # Check the trigger is on
            if self.triggerOn:
                self.getShot()
                theta = self.find_angle()

                # Compute rotation speed
                if theta is not None:
                    w = (theta/(50*(10**-3))) * 60/360
                    self.msgBox.setText("Rotation speed: "+str(w))
                    self.msgBox.exec()
                else:
                    self.msgBox.setText("No angle found")
                    self.msgBox.exec()
            else:
                self.msgBox.setText("Trigger is off")
                self.msgBox.exec()


# 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_())
```

We use the **findAngle.py** to compute all the functions related to the angle's calculation:

```
import numpy as np
import cv2
import math
import pprint

```

```
def houghcircles(npImage):

    # TODO ask parameters in a dialog
    npImage = cv2.cvtColor(npImage, cv2.COLOR_BGR2GRAY)
    circles = cv2.HoughCircles(npImage, cv2.HOUGH_GRADIENT, dp=1.2,
        minDist=20,
        param1=60, param2=40, minRadius=430, maxRadius=434)

    return circles[0][0] if len(circles) == 1 else None

def distColorRGB(img):
    """Make binary image by testing if pixel color is close to a
       selected RGB color.
     Distance is measured for each pixel p with color (r,g,b) to
       selected color (R,G,B) as
        d = max(abs(r-R), abs(g-G), abs(b-B)),
     where d is pixel value for resulting gray scale image
    """
    rgb = [160, 35, 35]           # Define the color to be detected
    A = img.astype(np.float32)    # Convert image to float32

    if (len(A.shape) > 2) and (A.shape[2] >= 3):         # Check if
       image is color
        B = np.ones(shape=A.shape, dtype=np.float32)     # Create an
            image with the color to be detected

        B[:, :, 0] = rgb[0]                              # Set the
           red channel
        B[:, :, 1] = rgb[1]                              # Set the
           green channel
        B[:, :, 2] = rgb[2]                              # Set the
           blue channel

        B = cv2.cvtColor(B, cv2.COLOR_BGR2RGB)          # Convert to
            gray scale

        D = np.max(np.abs(A-B),axis=2).astype(np.uint8)  # The
           distance image

    cv2.namedWindow("output", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("output", 800, 480)
    cv2.imshow('output', D)

    k = cv2.waitKey(0)
    if k == 27:          # wait for ESC key to exit
        cv2.destroyAllWindows()

    return D

def compute_angle(start_pt, end_pt, center_pt, im):
```

```
    # Compute the angle
    theta = math.degrees(math.atan2(end_pt[1] - center_pt[1],
       end_pt[0] - center_pt[0]) + 360) % 360

    # Draw the lines
    im = cv2.line(im, (int(center_pt[0]), int(center_pt[1])), (int(
       end_pt[0]), int(end_pt[1])), (255, 0, 0), 2)
    im = cv2.line(im, (int(center_pt[0]), int(center_pt[1])), (int(
       start_pt[0]), int(start_pt[1])), (255, 0, 0), 2)

    return theta, im

def centroid_interest_region(im):
    """Find the centroid of the interest region."""

    valid_x = []
    valid_y = []

    # Find the distance image for the interesting color
    binaryIm = distColorRGB(im)

    # Check which pixels are in the interesting region
    for y in range(im.shape[0]):
        for x in range(im.shape[1]):
            if binaryIm[y, x] < 10:
                valid_x.append(x)
                valid_y.append(y)

    # Calculate the centroid of the interesting region
    mx = np.mean(valid_x)
    my = np.mean(valid_y)

    return mx, my

def findAngle(start_im: str, end_im: str):

    # Find the starting point
    start_pt = centroid_interest_region(start_im)

    # Find the ending point
    end_pt = centroid_interest_region(end_im)

    # Find the center of the circle
    circle = houghcircles(end_im)

    # Find the angle
    if circle is not None:

        theta, angle_im = compute_angle(start_pt, end_pt, (circle
```

```
            [0], circle[1]), end_im)

        # Print information
        """
        print('Start point: ', start_pt)
        print('End point: ', end_pt)
        print('Center point: ', circle)
        print('Angle: ', angle)
        """
    else:
        print('No circle found')
        theta = None
        angle_im = None

    return theta, angle_im


if __name__ == '__main__':

    # Read the initial image to find the starting point
    start_im = cv2.imread('.\\appImageViewer4J\images\
       Initial_angle_disk.jpg')

    # Read the image to find the ending point
    end_im = cv2.imread('.\\appImageViewer4J\images\\50
       ms_disk_speedup.jpg')

    # Find the angle
    findAngle(start_im, end_im)
```

5 Time table

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