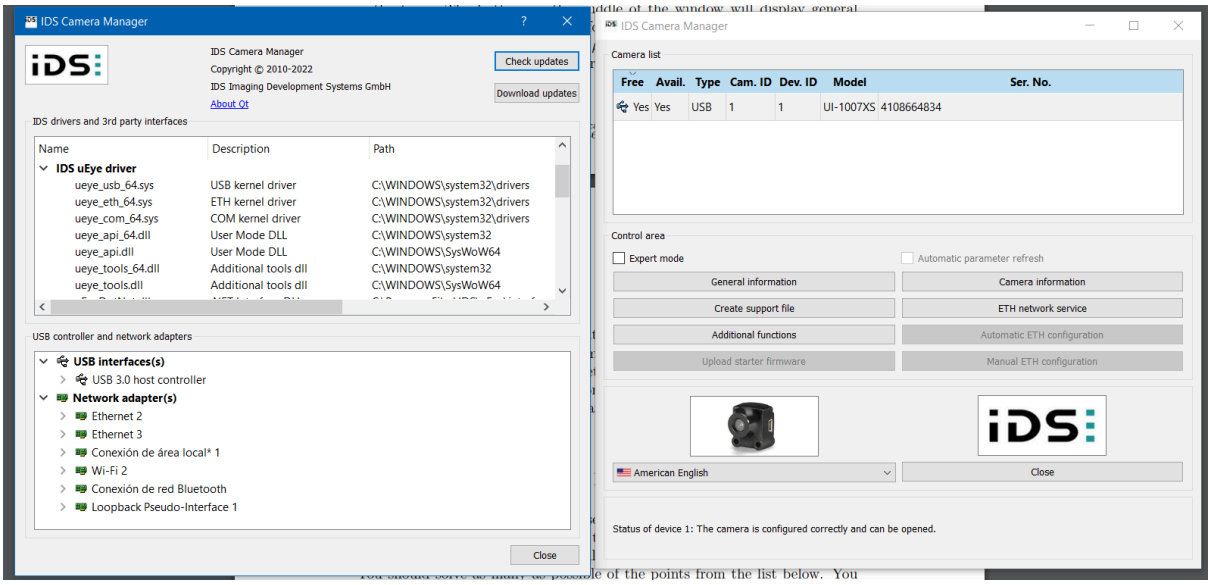
Image Acquisition Assignment 3

Antón Maestre Gómez and Daniel Linfon Ye Liu

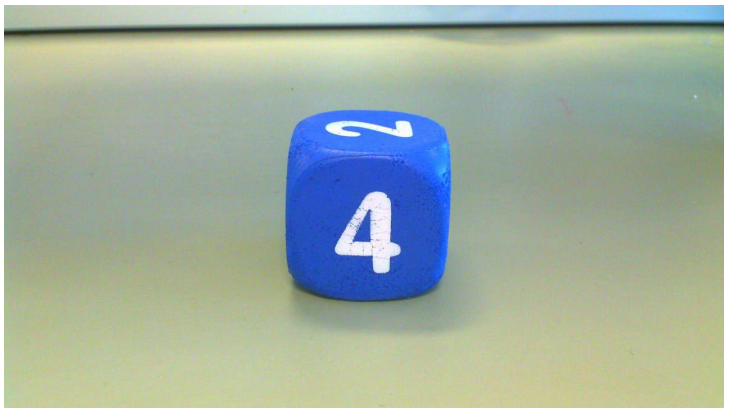
3 Image acquisition using IDS camera

3.1 Use IDS programs

We have started "IDS Camera Manager":



General information

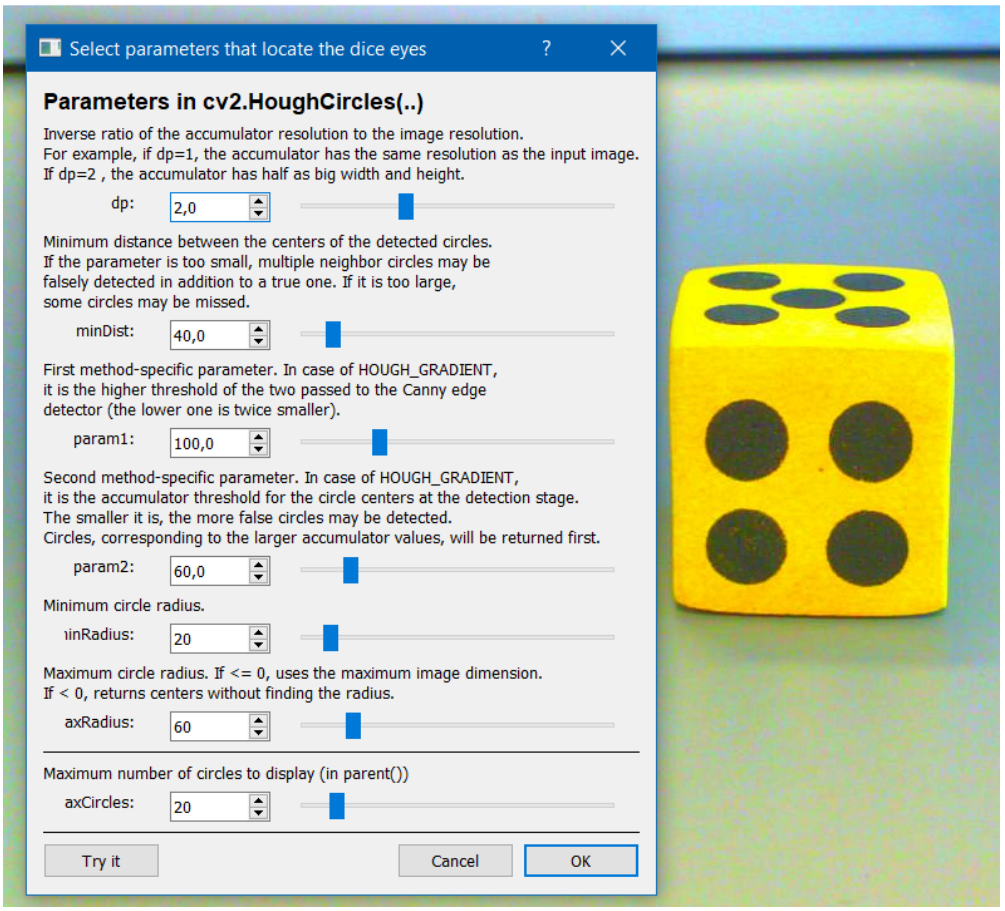


Exposure: We have increased the exposure time to get more light.
Saturation: We have lowered the saturation level

Exposure: We have decreased the exposure time to avoid overexposure.
Saturation: We have increased the saturation level to highlight the yellow.

3.2 Use IDS camera and Python

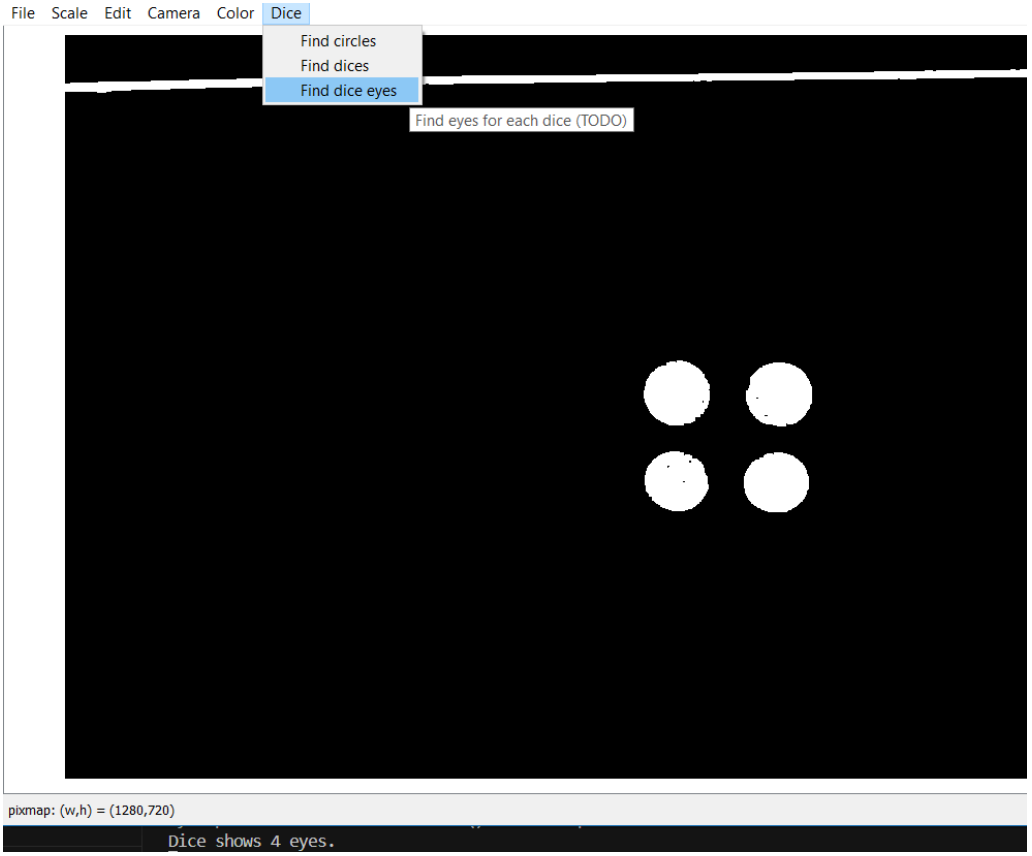
a) Add a feature (an action) to the Dice menu that displays a dialog window where you select parameters to the cv2.HoughCircles() function, try this, and show the effects. Note that this is done in the example file appImageViewer3.py under the Dice menu, and you may copy code from this file and use clsHoughCirclesDialog.py, as long as you understand what is done.



b) Add a feature (an action) to the Dice menu that finds the number of eyes in a dice in the captured image. The results may be printed to standard output or shown on image. Start simple with only one dice in the image, and color does not matter. Example: Dice shows 3 eyes.

To count the eyes of the dice, we have used the same logic as the black dots function, done in the previous task. We have detected the white contours and filtered them so that they meet certain characteristics of the circles (area, perimeter, bounding box...). If they meet them, we will assume that it is a circle and we will increment a counter.

```
def findEyes(self):
    """Find how many eyes each dice has using ??."""
    # To gray
    if (len(self.npImage.shape) == 3) and (self.npImage.shape[2] >= 3):
        self.prevPixmap = self.pixmap
        if (self.npImage.shape[2] == 3):
            gray_image = cv2.cvtColor(self.npImage, cv2.COLOR_BGR2GRAY)
        if (self.npImage.shape[2] == 4):
            gray_image = cv2.cvtColor(self.npImage, cv2.COLOR_BGRA2GRAY)

    # Thresholding to binary
    _, binary_image = cv2.threshold(gray_image, 90, 255, cv2.THRESH_BINARY_INV)

    # Apply morphological operations to remove noise
    kernel = np.ones((3, 3), np.uint8)
    clean_image = cv2.morphologyEx(binary_image, cv2.MORPH_OPEN, kernel, iterations=2)
    self.np2image2pixmap(clean_image, numpyAlso=True)

    # Detect contours
    contours, _ = cv2.findContours(clean_image, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

    # Filter out contours that are circular in shape
    self.eyes_count = 0
    for contour in contours:
        area = cv2.contourArea(contour)
        perimeter = cv2.arcLength(contour, True)

        # Calculate bounding box to avoid other figures like lines
        x, y, w, h = cv2.boundingRect(contour)
        aspect_ratio = w / float(h)

        if 0.75 < aspect_ratio < 1.2:  # Adjust for more or less circular eyes
            if perimeter > 0:
                circularity = 4 * np.pi * (area / (perimeter ** 2))
                if 0.7 < circularity < 1.3:  # Approximate circularity for a circle
                    self.eyes_count += 1

    print(f"Dice shows {self.eyes_count} eyes.")
```



c) Find the color of the dice. This can be done in many ways, perhaps considering pixels just outside the radius for each circle that locate the black dot (dice eye). You should consider a table with color names and values, and allow for a small range around each color.

The logic used to do this activity is the following:

1. Using the cv2.HoughCircles function, we detect the circles of the dice and extract their x and y coordinates. We also extract their radius.

2. After getting that data, we look for the pixels that are around each circle in a range of 1.2 times the radius.

3. When we get all the pixels within that range, we calculate the average of their colors in the RGB channel.

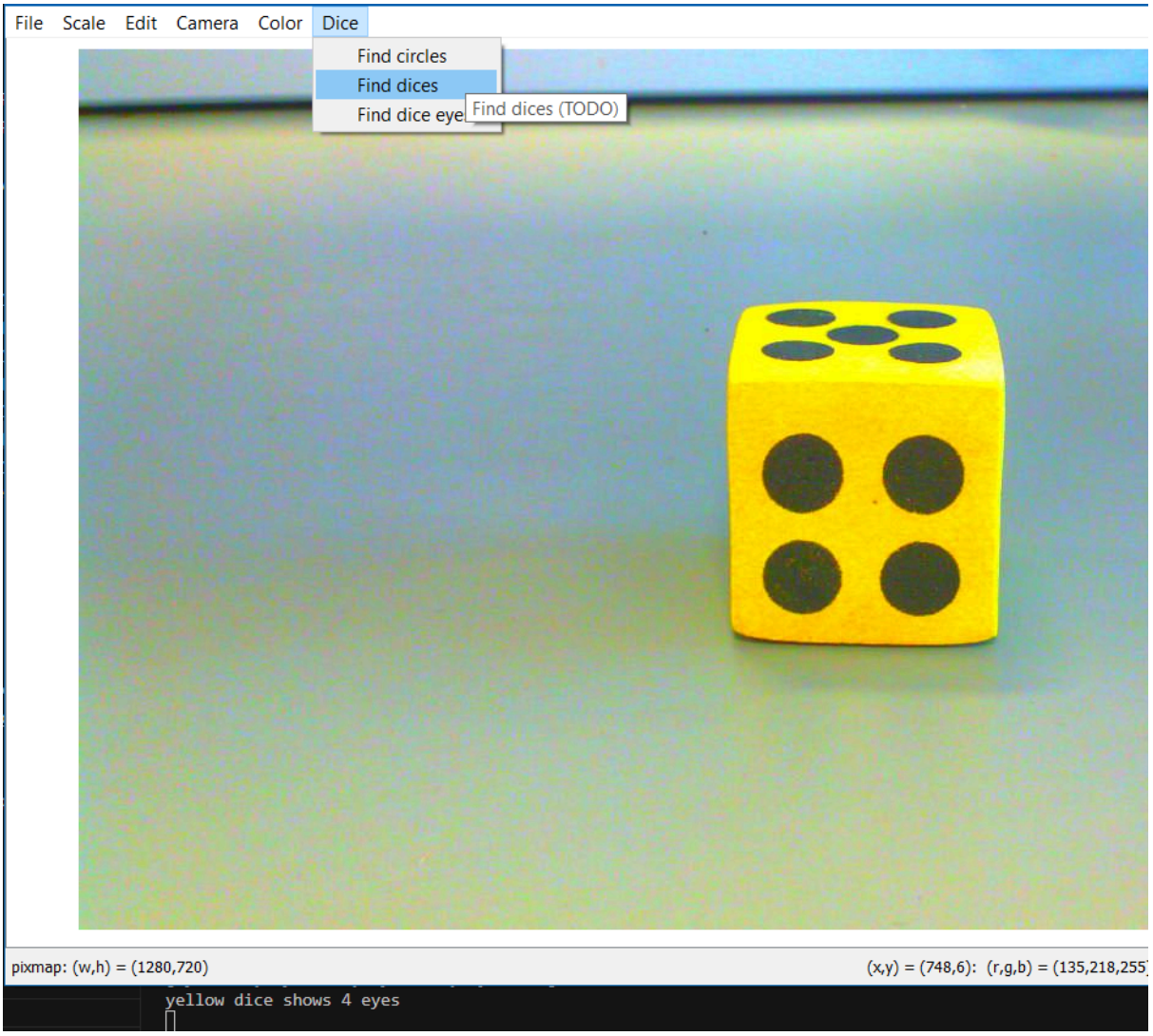
4. Finally, we observe which color that average corresponds to approximately and we add it to the results list. In this list, there will be as many colors as there are eyes in the image. For example, in our image of the yellow die, the yellow color will appear 4 times:

```
['yellow', 'yellow', 'yellow', 'yellow']
```

We will convert that list into a dictionary and thus we will have the color of the die with its number of eyes.

```python
def findDices(self):
    """Find dices in active image using ??."""

    # We create a copy of the original image to convert it into gray and detect the circles
    gray_image = self.npImage
    # To gray
    gray_image = cv2.cvtColor(gray_image, cv2.COLOR_BGR2GRAY)
    hsv_image = cv2.cvtColor(self.npImage, cv2.COLOR_BGR2HSV)
    d = HoughCirclesDialog(self, title="Select parameters that locate the dice eyes")
    (dp, minDist, param1, param2, minRadius, maxRadius, maxCircles) = d.getValues()   # display dialog and return values
    if d.result():
        C = cv2.HoughCircles(gray_image, cv2.HOUGH_GRADIENT, dp=dp, minDist=minDist,
                param1=param1, param2=param2, minRadius=minRadius, maxRadius=maxRadius)


    self.eyes_count = 0
    if C is not None:
            C = np.int16(np.around(C))
            #print(f"  'C'  is ndarray of {C.dtype.name}, shape: {str(C.shape)}")
            results = []
            for i in range(min(maxCircles, C.shape[1])):
                (x,y,r) = ( C[0,i,0], C[0,i,1], C[0,i,2] )  # center and radius

                # Define a range of points outside the circle to analyze the color
                outer_radius = int(1.2 * r)  # 1.2 times circle's radious

                # List to store the colors of pixels outside the circle
                colors = []

                # Extract points around the circle at 1.2 times the radius
                for angle in range(0, 360, 10):  # Take samples every 10 degrees
                    angle_rad = np.deg2rad(angle)
                    x_outside = int(x + outer_radius * np.cos(angle_rad))
                    y_outside = int(y + outer_radius * np.sin(angle_rad))

                    # Get the color of the pixel outside the circle in space HSV
                    colors.append(hsv_image[y_outside, x_outside])


                # Average the colors obtained
                if colors:
                    colors = np.array(colors)
                    avg_color = np.mean(colors, axis=0).astype(int)

                    # Determine the color of the die based on the average color
                    color_name = self.getColorName(avg_color)
                    results.append(color_name)
                    #print(f"Detected dice color: {color_name} at position ({x}, {y})")


                # Draw the circles in the original image
                cv2.circle(gray_image, (x,y), r, (255, 0, 255), 3) # and circle outline
                self.eyes_count += 1

    self.np2image2pixmap(self.npImage, numpyAlso=True)

    # Color with the most frecuency
    print(results)
    count = Counter(results)
    color_count = dict(count)

    for color, eyes_count in color_count.items():
        print(f"{color} dice shows {eyes_count} eyes")
```

```python
def getColorName(self, hsv_color):
    """Determines the color name based on the average HSV values."""

    # Color table
    COLOR_TABLE_HSV = {
        "blue": [(100, 150, 50), (130, 255, 255)],
        "yellow": [(20, 100, 100), (30, 255, 255)],
        "red": [(0, 100, 100), (10, 255, 255)],
        "green": [(40, 100, 100), (70, 255, 255)],
        "white": [(0, 0, 200), (180, 50, 255)],
        "black": [(0, 0, 0), (180, 255, 50)]
    }

    # Compare the average HSV color to the color table and return the corresponding name
    for color_name, (lower_bound, upper_bound) in COLOR_TABLE_HSV.items():
        if all(lower <= hsv_color[i] <= upper for i, (lower, upper) in enumerate(zip(lower_bound, upper_bound))):
            return color_name

    return "unknown"  # Not found
```



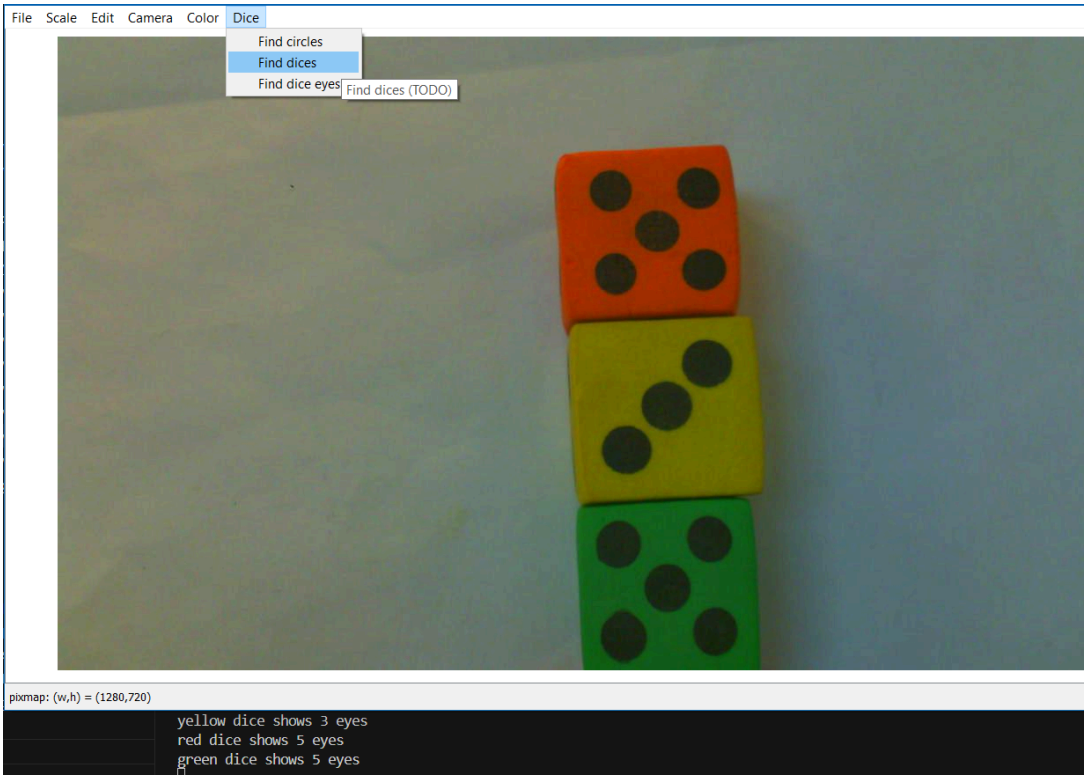
d) Print results for an image with one dice. Start simple with only one dice in the image.

Using the dictionary, we have the color of the dice with its number of eyes

```
# Color with the most frecuency
count = Counter(results)
color_count = dict(count)

for color, eyes_count in color_count.items():
    print(f"{color} dice shows {eyes_count} eyes")
```

e) Print results for an image with two or more dices.

In the list we will have different colors, each color means a dice and the number of times it is repeated will be the number of eyes on that dice. Therefore, when transforming the list into a dictionary, we will have each die with its number of eyes.



f) Add a feature (an action) to the Camera menu that displays a dialog window where you can change (at least one) camera option. This dialog should be considerably simpler than the dialog in the µEye Cockpit program.

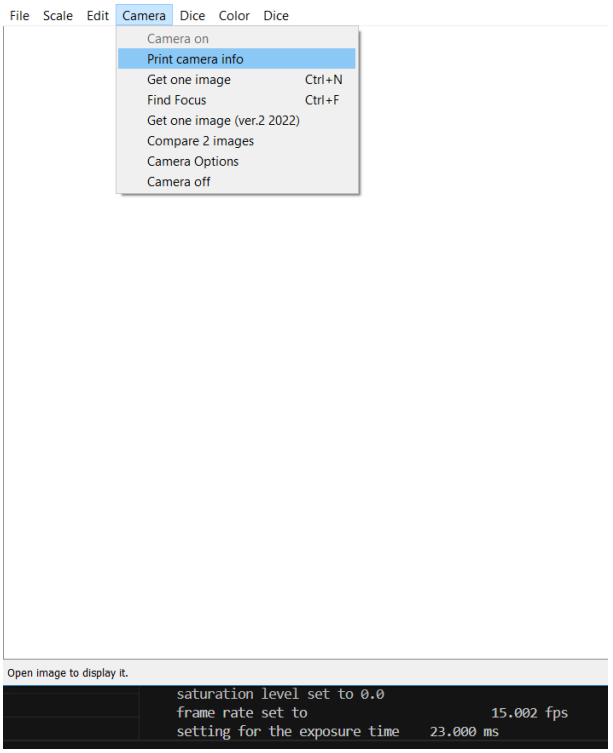
To display a dialog, we have created a new class:

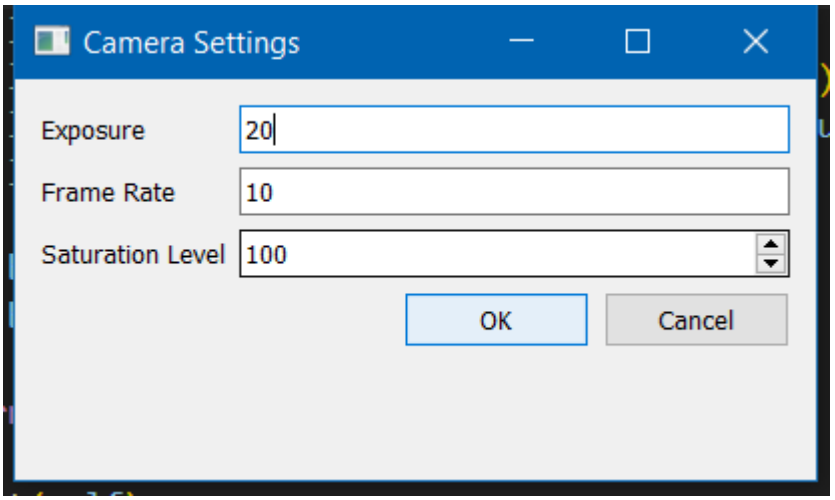
```
class CamOpt_dialog(QWidget):
    """Display a dialog that asks for few parameters
    to thath will be returned
    """
    def __init__(self, mainApp):
        super().__init__()
        super().setWindowTitle("Camera Settings")
        super().resize(400, 200)

        # To call the mainApp methods
        self.mainApp = mainApp

        self.exposure = QLineEdit(self)
        self.frameRate = QLineEdit(self)

        self.saturation = QSpinBox(self)
        self.saturation.setRange(-100, 155)
        self.saturation.setValue(0)  # Valor inicial predeterminado

        self.buttonBox = QDialogButtonBox(QDialogButtonBox.Ok | QDialogButtonBox.Cancel, self)

        self.layout = QFormLayout(self)
        self.layout.addRow("Exposure", self.exposure)
        self.layout.addRow("Frame Rate", self.frameRate)
        self.layout.addRow("Saturation Level", self.saturation)
        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.exposure.text(),
            self.frameRate.text(),
            self.saturation.value()
        )

        self.mainApp.changeOptions(returnedValue)

        return

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

        return
```

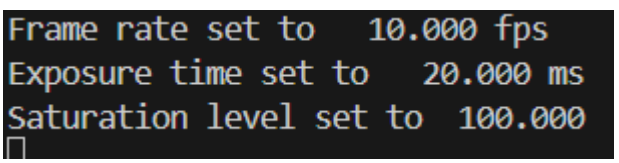
Being a new feature of the camera section, we have created these functions in our previous script appImageViewer2P.py

```python
# A3

def cameraOptions(self):
    """Display the camera options dialog"""
    if self.camOn:
        self.cameraOps.show()

    return

def changeOptions(self, options):
    """Change the camera options"""
    # Get the input from the user
    exposure, frameRate, saturation = options

    d = ueye.DOUBLE()  # Variable to capture the result

    # Frame rate: Set frame rate and check the return value
    try:
        self.frameRate = ueye.DOUBLE(float(frameRate))  # Convert to float
        retVal = ueye.is_SetFrameRate(self.cam.handle(), self.frameRate, d)
        if retVal == ueye.IS_SUCCESS:
            print(f"Frame rate set to {float(self.frameRate):8.3f} fps")
        else:
            print("Failed to set frame rate")
    except ValueError:
        print("Invalid frame rate value")

    # Exposure: Set exposure time and check the return value
    try:
        self.exposure = ueye.DOUBLE(int(exposure))  # Convert to float
        retVal = ueye.is_Exposure(self.cam.handle(),
                                  ueye.IS_EXPOSURE_CMD_SET_EXPOSURE,
                                  self.exposure, ueye.sizeof(self.exposure))
        if retVal == ueye.IS_SUCCESS:
            print(f"Exposure time set to {float(self.exposure):8.3f} ms")
        else:
            print("Failed to set exposure time")
    except ValueError:
        print("Invalid exposure value")

    # Saturation: Set saturation level and check the return value
    try:
        self.saturation = ueye.DOUBLE(int(saturation))  # Convert to int
        retVal = ueye.is_Saturation(self.cam.handle(),
                                    ueye.SATURATION_CMD_SET_VALUE,
                                    self.saturation, 4)
        if retVal == ueye.IS_SUCCESS:
            print(f"Saturation level set to {float(self.saturation):8.3f}")
        else:
            print("Failed to set saturation level")
    except ValueError:
        print("Invalid saturation value")
```

We have added the "Camera Options" option to the camera menu, which displays the created window. We have added the ability to change saturation, exposure and frame rate.







g) **The final task here is to add yet another action to the Dice menu that captures video and continuously finds and shows (print) the number of eyes in each dice in the video scene.**

For this task, we have downloaded from the internet the "SimpleLive_Pyueye_OpenCV.py" script that allows recording a video:

https://sanxo.eu/simple-live-image-acquisition-with-the-python-interface-pyueye/

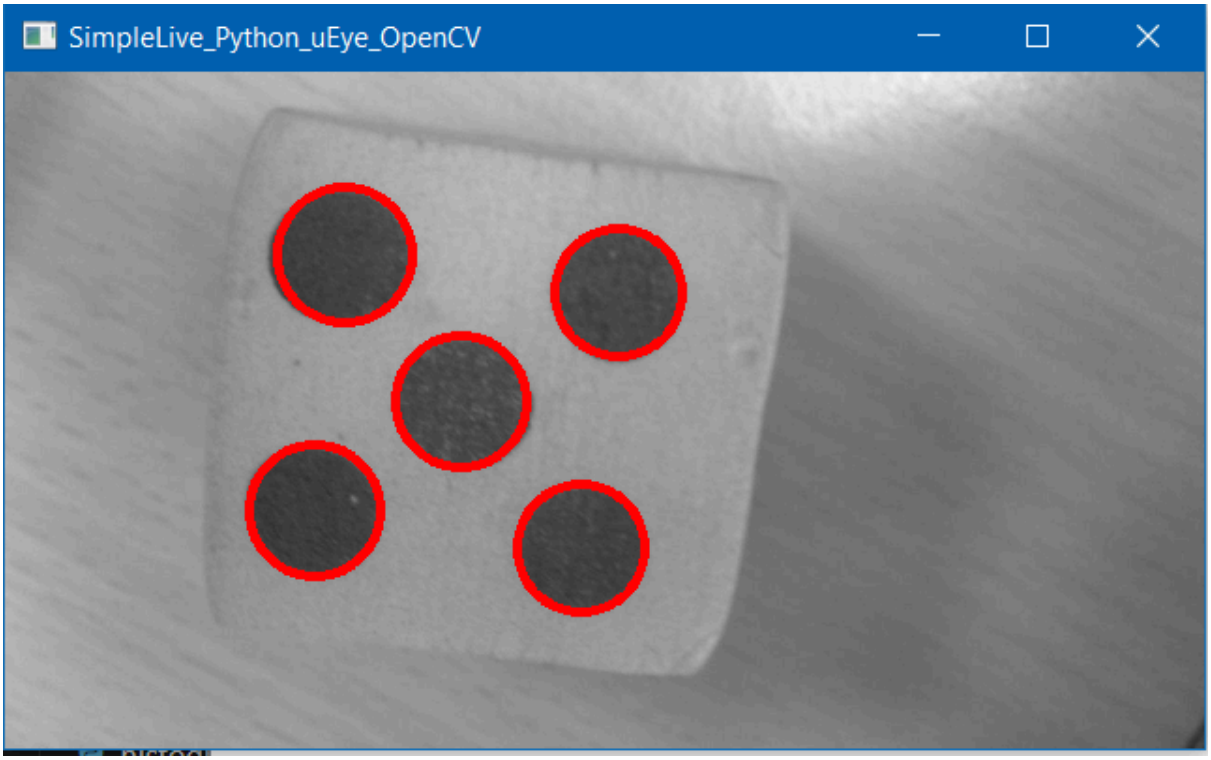
Based on this, we have modified it by adding the functionality of our find circles function.

```
#Include image data processing here
if process:
    frame = cv2.cvtColor(frame, cv2.COLOR_BGR2GRAY)

    circles = cv2.HoughCircles(frame, cv2.HOUGH_GRADIENT, dp=2, minDist=40, param1=100, param2=60, minRadius=20, maxRadius=60)

    frame = cv2.cvtColor(frame, cv2.COLOR_GRAY2BGR)

    list_number_circles=[]
    if circles is not None:
        circles = np.uint16(np.around(circles))
        for i in range(min(18, circles.shape[1])):
            (x,y,r) = ( circles[0,i,0], circles[0,i,1], circles[0,i,2] )  # center and radius
            # Draw the circles in the origina image
            cv2.circle(frame, (x,y), r, (0, 0, 255), 3) # and circle outline
            list_number_circles.append(len(circles))

        number_circles= sum(list_number_circles)

    else:
        pass

cv2.imshow("SimpleLive_Python_uEye_OpenCV", frame)
```

record_video function from SimpleLive_Pyueye_OpenCV.py

```
def findDices_Video(self):

    if self.camOn:
        self.cameraOff()

    record_video(process=True)

    return
```

From appImageViewer3P.py



You can't see it in the document, but if you move the dice, the circles also move.

Working hours

Date	Time	Student working
September 12	LAB: 10:00 - 15:00	Daniel & Anton
September 13	LAB: 10:00 - 12:30	Daniel & Anton
September 13	HOME: 17:00 - 21:00	Daniel & Anton