{Learn, Create, Innovate};

Computer Vision

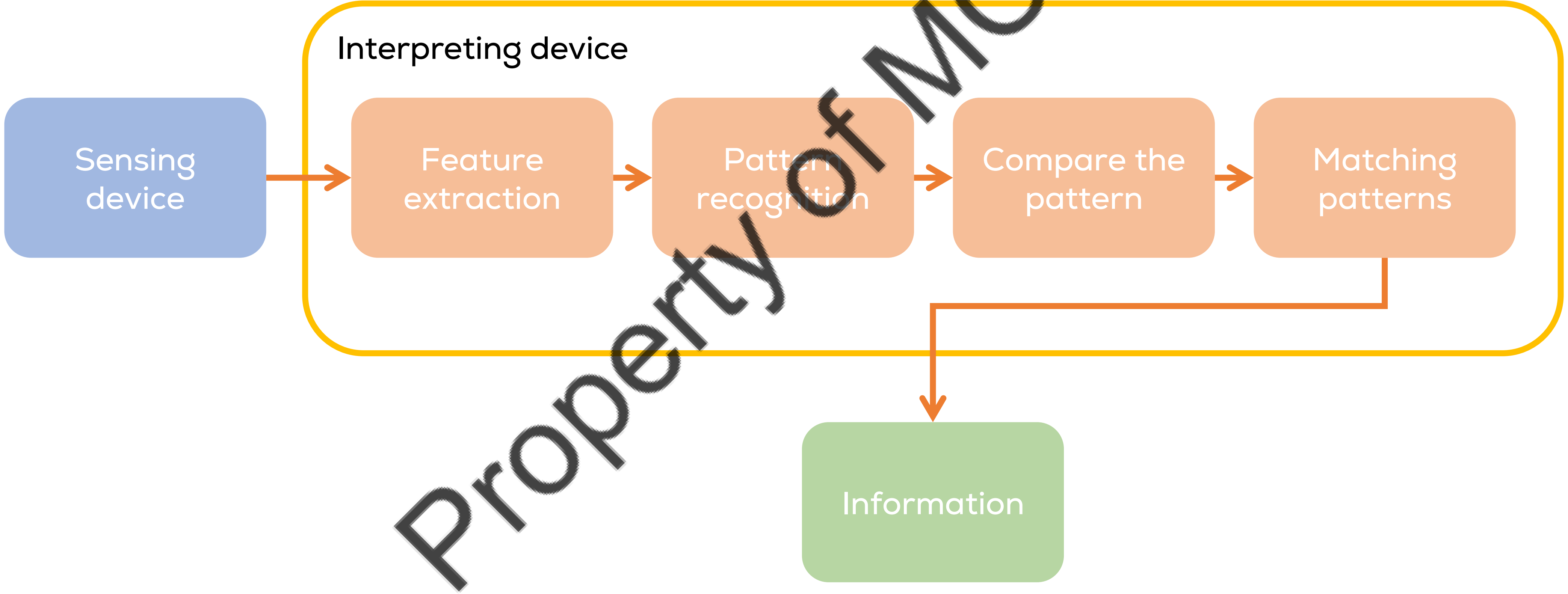
OpenCV

MCR2 Manchester **Robotics**

Computer Vision

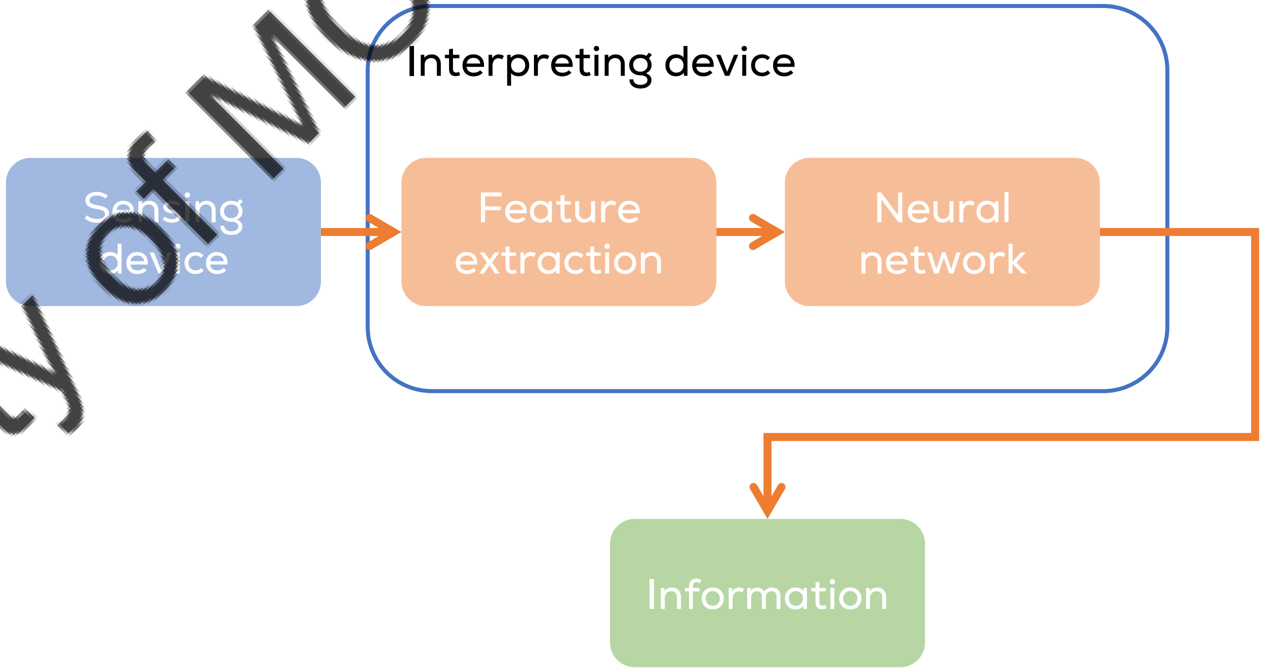
- A sub field in computer science and artificial intelligence dedicated to identify and understand features in an image or video.
- Artificial intelligence mimics the thinking process. Computer vision aims to reproduce human sight and inference.
- Many modern learning-based techniques use computer vision as an entry node to perform inference methods:
    - Artificial intelligence, machine learning, and deep learning

Computer Vision General Pipeline



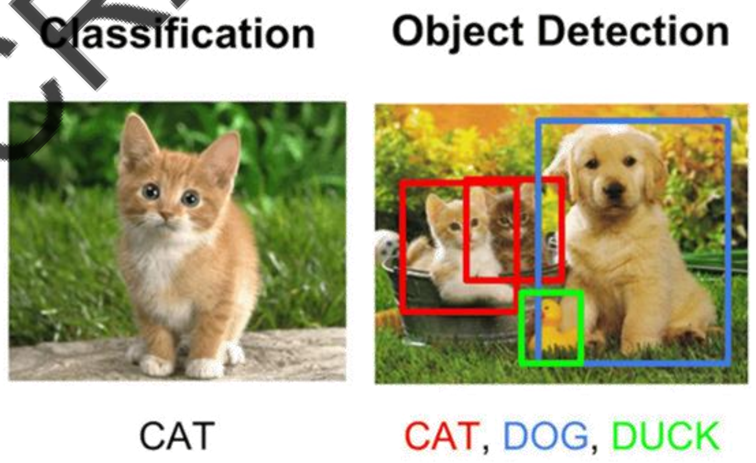
Computer vision with deep learning

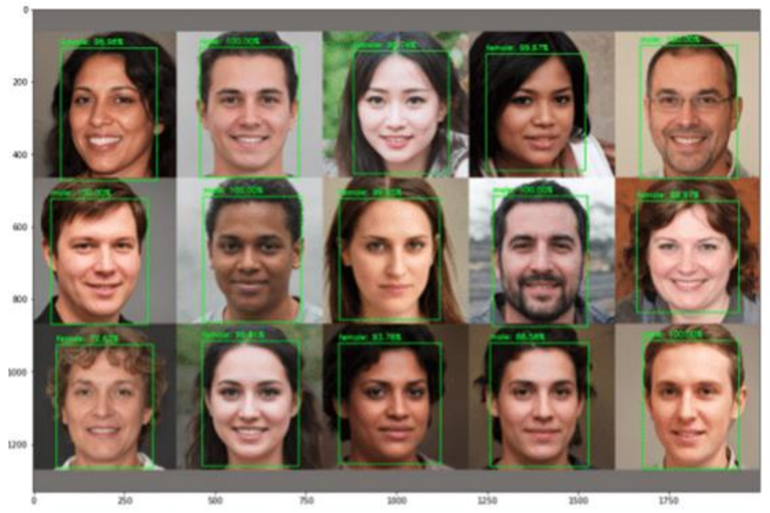
- State-of-the-art techniques moving from statistical and mathematical transformations for analyzing images to pixel-by-pixel analysis.
- *Deep learning (DL)* requires large amounts of data for it to learn about the context of visual data.
  - After several iterations, the model "learns" to differentiate image features.
- *CNN* decomposes the image into pixels that are labeled.
  - It uses convolutions to predict the tag according to its input pixel.
  - The accuracy of the prediction should increase with time.



Computer vision capabilities

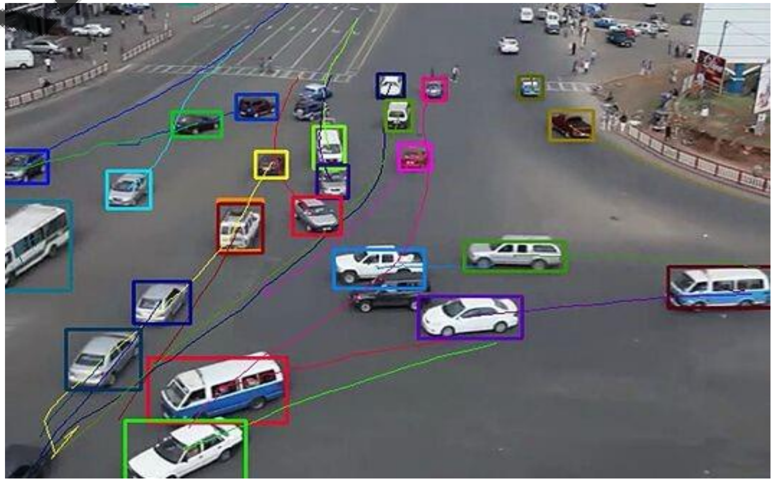
- Object classification
  - Identifies/categorizes what is in the image (e.g., cat, dog).
  - Output: A label or class for the entire image.
  - Example: "This image contains a cat“
  - Usage: Image search, simple classification, content tagging
- Object identification
  - Identifies characteristics and what and where objects are in the image.
  - Outputs: Bounding boxes + labels for each object found
  - Example: "There’s a cat at (x1, y1, x2, y2) and a dog at…“
  - Usage: Autonomous driving, security cameras, counting objects.

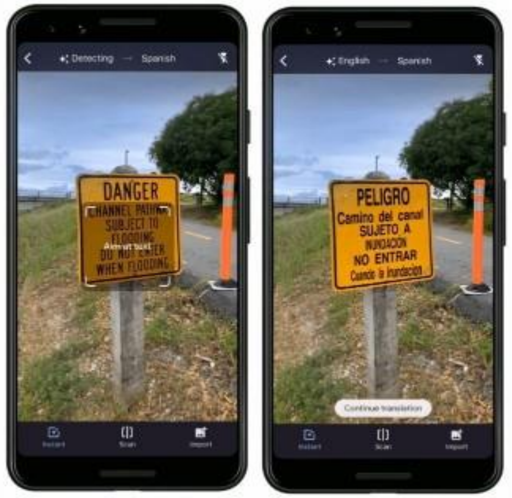




Computer vision capabilities

- Object tracking
  - Follows/ Process the location of a specific object across video frames
  - Input: A video (or sequential frames) with moving objects
  - Output: Path/coordinates of the object over time
  - Example: Track a car across security camera footage
  - Usage: Surveillance, sports analytics, autonomous vehicles
- Optical character recognition
  - Letters and numbers identification to convert it into a set of machine-encoded text
  - Input: Text extracted from the image
  - Output: A photo, scanned document, or frame with text
  - Example: Read the text from a street sign or a scanned book page
  - Usage: Document digitization, license plate recognition, translations




Detecting → Spanish
English → Spanish
DANGER
CHANNEL PATH SUBJECT TO FLOODING DO NOT ENTER WHEN FLOODING
PELIGRO
Camino del canal SUJETO A INUNDACION NO ENTRAR Cuando la inundación
Continue translation

OpenCV

- Open-Source Computer Vision Library
- Cross-platform, free to use
- Originally developed by Intel
- Aimed at real-time computer vision
- Contains a wealth of functions and routines for the real-time processing and analysis of images

OpenCV Installation (Python)

- Windows
  - Open a cmd prompt and use the following command (make sure you have pip installed):

```
pip install opencv-python
```

- Ubuntu
  - Open a terminal and use the following command:

```
sudo apt update
sudo apt install -y python3-opencv libopencv-dev
```

  - Please be aware that this is just a basic installation of OpenCV. If you require CUDA or other OpenCV packages such as Aruco, you must install it from source (ubuntu) or using pip in Windows. More information here and here

OpenCV Basics

- Once installed OpenCV, it is possible to start programming in Windows or Ubuntu.
- In this section, only the basics of OpenCV will be shown.
- In the next section, the student will learn to use ROS 2 with OpenCV.
- For the following examples, download the following images and videos from GitHub.

Puzzlebot
Jetson Edition

OpenCV

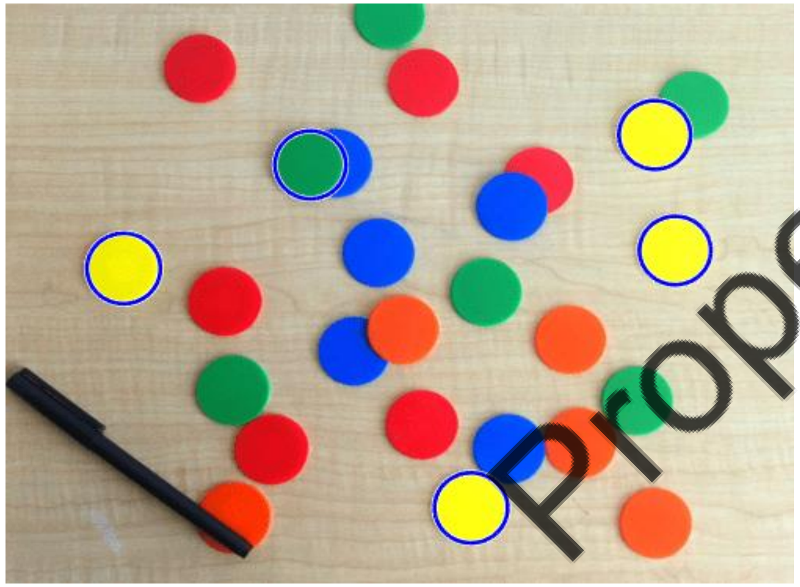
Colour Detection

{Learn, Create, Innovate};

MCR2 Manchester Robotics

Colour Detection: What is the problem?

- We aim to detect connected regions in the space with a characteristic shape and colour palette.

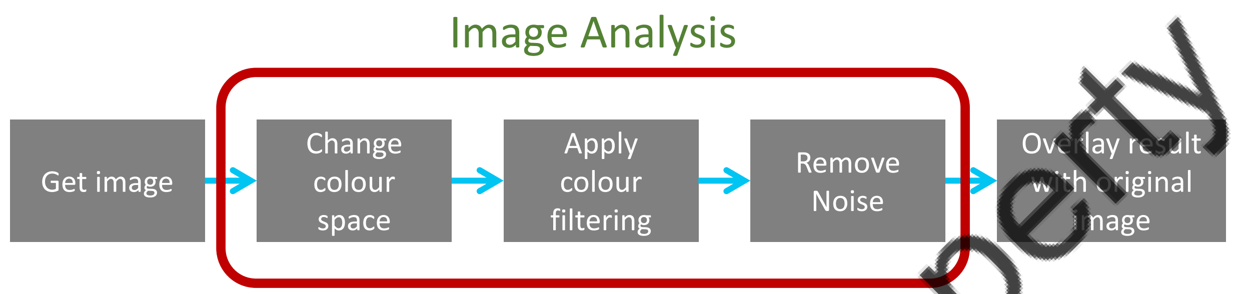

Example of the problem

Characteristics to exploit

1. We can look for patches of certain colours
2. We can use size to reject shapes
3. We can look for circular shapes in the image

Colour Detection

- The following image, shows a simple colour detection pipeline that can be implemented in OpenCV.

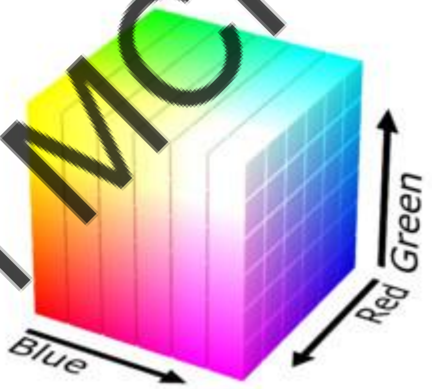


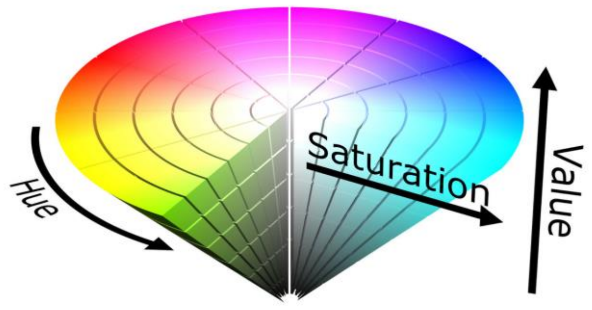
Colour Spaces

- OpenCV manages different colour representations (more than 150), called colour spaces.
- The most widely used are BGR (Blue, Green, Red), HSV (Hue, Saturation, Value) and Gray (Grayscale)
- OpenCV offers conversions amongst these spaces i.e., BGR ↔ Gray and BGR ↔ HSV, etc.
- More information here.

Colour Detection

- BGR (Blue, Green, Red):
  - Default colour format in OpenCV.
  - It is not ideal for colour detection due to the mixing of colour intensities.
- HSV (Hue, Saturation, Value):
  - Hue (H): Represents colour type (0 to 179 degrees in OpenCV).
  - Saturation (S): Concentration/diluting of colour. Unsaturated to represent shades of grey and fully saturated (no white component). (0 to 255)
  - Value (V): Brightness or intensity of colour (0 to 255).
- Why HSV?
  - More intuitive way to define/describe colours.
  - Used to perform colour-based segmentation





```
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)
```

Colour Detection

Colour Range Thresholding

- Now that the colour is on a more practical scale, colours can be filtered by range using the “inRange” function.
- The output is a binary mask.

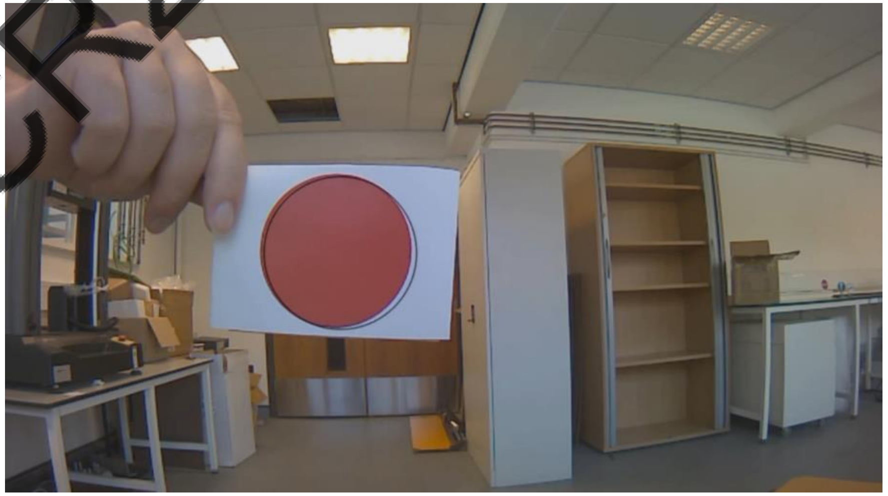
Basic Idea:

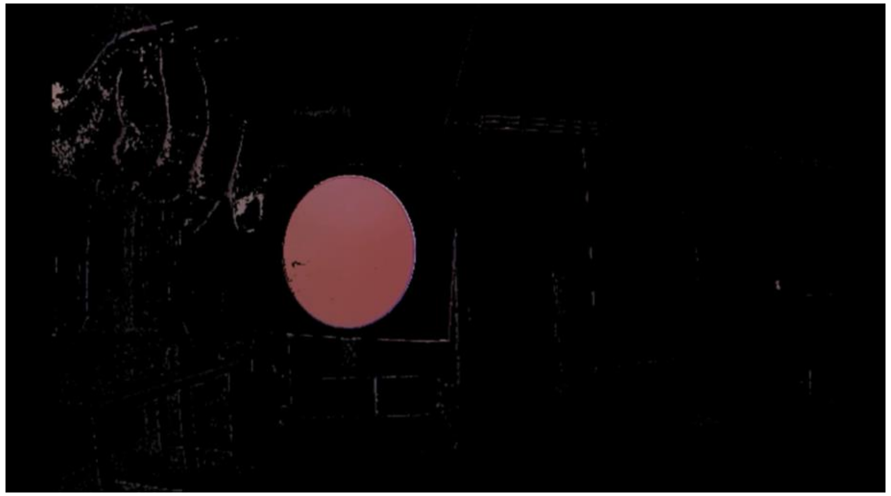
- Check if a pixel $[H, S, V]$ values are within a range i.e.,

$$[H_l, S_l, V_l] \leq [H, S, V] \leq [H_u, S_u, V_u]$$

- If the colour is in a range set the pixel to 255 (white)
- Else, set to 0.

```
#Blue Ranges
blue_lower_limit = np.array([90 90, 90])
blue_upper_limit = np.array([110,255,255])

#Blue Mask
blue_mask = cv.inRange(hsv_image, blue_lower_limit, blue_upper_limit)
```

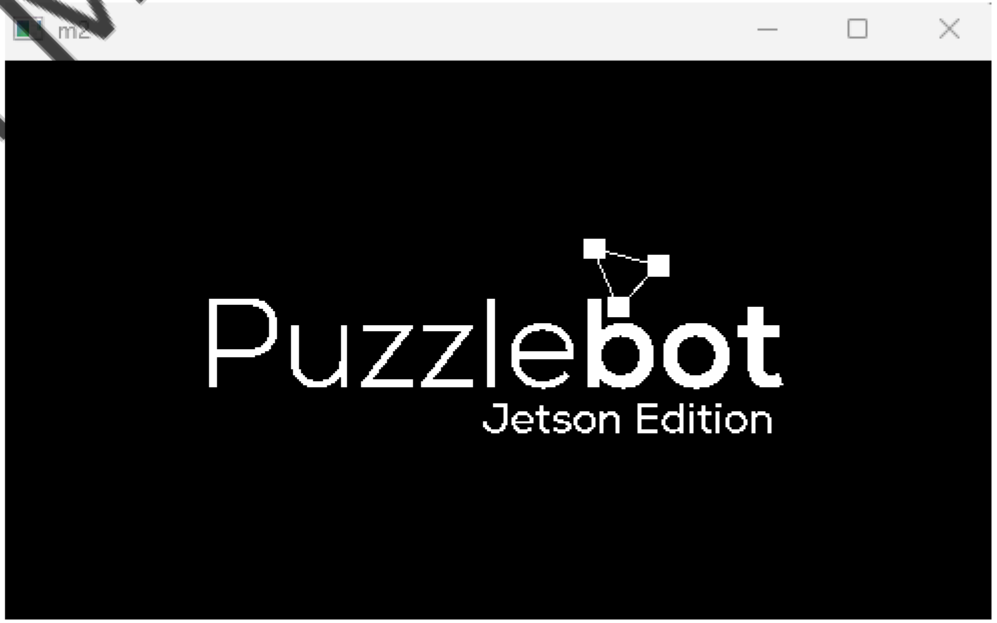




Thresholding, Masks and Bitwise Operations

Masks

- A binary image (black & white) used to select parts of another image.
- Usually:
  - White (255): Areas to keep/process.
  - Black (0): Areas to ignore/mask out.
- Example Usage:
  - Focus operations on areas of interest.
  - Create effects and isolate objects.
  - Usually obtained by using the "inRange" function.



Thresholding, Masks and Bitwise Operations

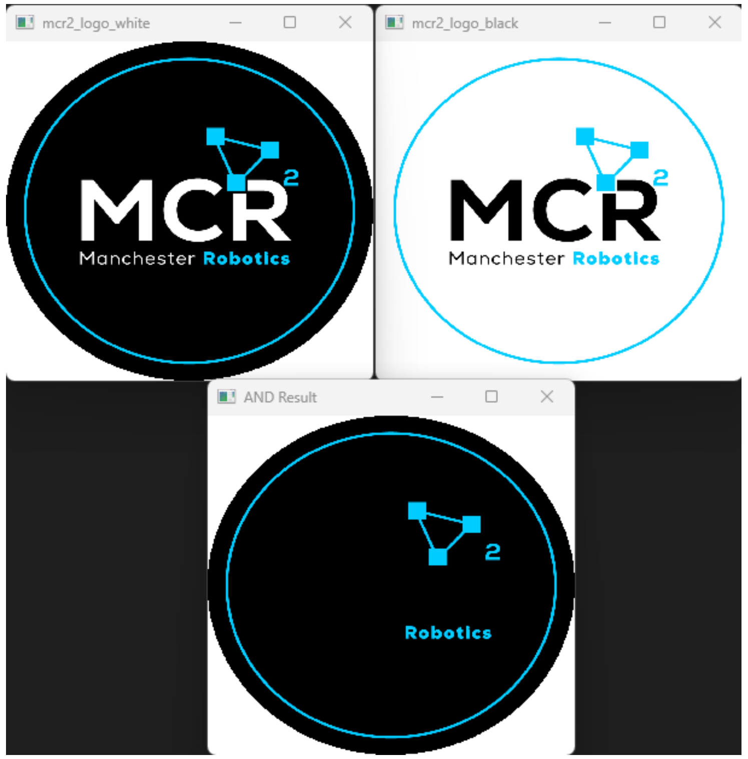
Bitwise Operations: AND

- Takes two images (src1 and src2) and performs an AND operation between corresponding pixels.
- If both pixel values are non-zero (i.e., 1), the result is non-zero (i.e., 1). Otherwise, it's zero.
- AND Example:

```
Pixel A --> 11101010 (binary)
Pixel B --> 01010101 (binary)
-------------------------------
Result --> 01000000
```

- In OpenCV, these operations are typically used to apply a mask to preserve specific areas of an image.
- Extract parts of an image that satisfy certain conditions.

```
result = cv2.bitwise_and(image, image, mask=mask)

## If mask used Only pixels where mask is white (255) will be
visible in result.The black parts of the mask (0) hide pixels
in the output..
```



Thresholding, Masks and Bitwise Operations

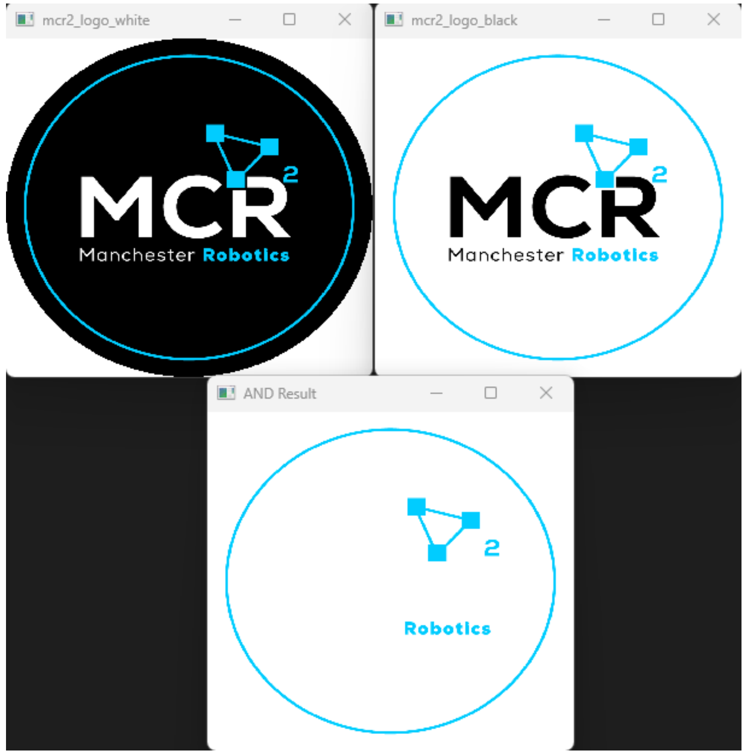
```
result = cv2.bitwise_or(img1, img2)
```

Bitwise Operations: OR

- Performs an OR operation between two images.
- If either pixel is non-zero, the result is non-zero.
- OR Example:

```
Pixel A --> 0 0 1 0 1 0 1 0 (binary)
Pixel B --> 0 1 0 1 0 0 0 0 (binary)
--------------------------------
Result --> 0 1 1 1 1 0 1 0
```

- In OpenCV, these operations are typically used to Combine two images or regions.



Thresholding, Masks and Bitwise Operations

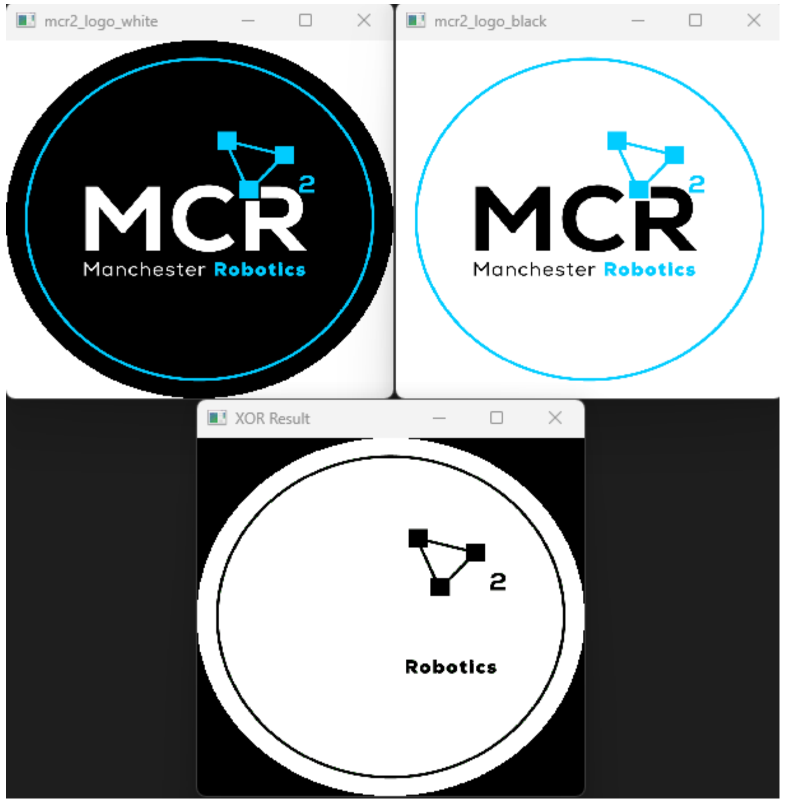
Bitwise Operations: XOR

- Performs an XOR operation. If only one of the pixels is non-zero, the result is non-zero.
- If both are zero or both are non-zero, the result is zero.
- XOR Example:

```
Pixel A --> 0 0 1 0 1 0 1 0 (binary)
Pixel B --> 0 1 0 1 0 0 1 0 (binary)
--------------------------------
Result --> 0  1  1  1  1 0 0 0
```

- In OpenCV, these operations are typically used to Highlight differences between two images or regions.

```
result = cv2.bitwise_xor(img1, img2)
```



Thresholding, Masks and Bitwise Operations

Bitwise Operations: NOT

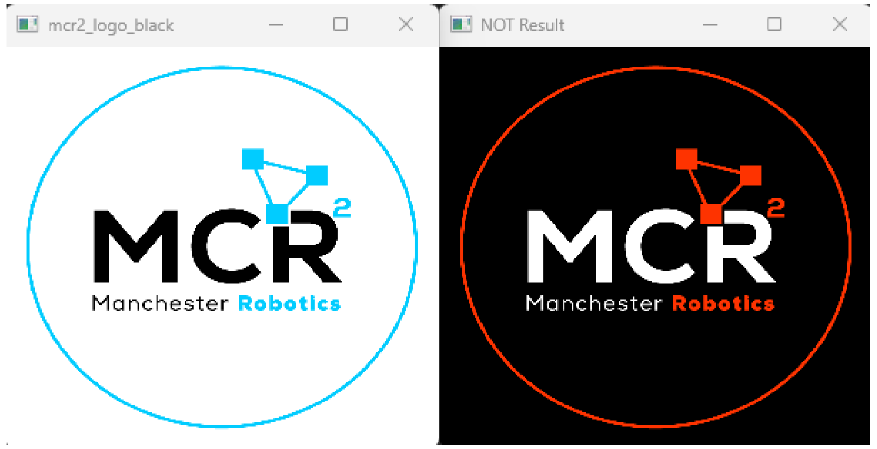
- Inverts every bit of the image.
- Converts white pixels (255) to black (0) and vice versa.
- NOT Example:

Pixel A --> 0 0 1 0 1 0 1 0 (binary)

Result --> 1 1 0 1 0 1 0 1

- In OpenCV, these operations are typically used invert a mask or an image.

```
result = cv2.bitwise_not(img1)
```



Activity 1

Colour Filtering

{Learn, Create, Innovate};

MCR2 Manchester Robotics

Colour Filtering

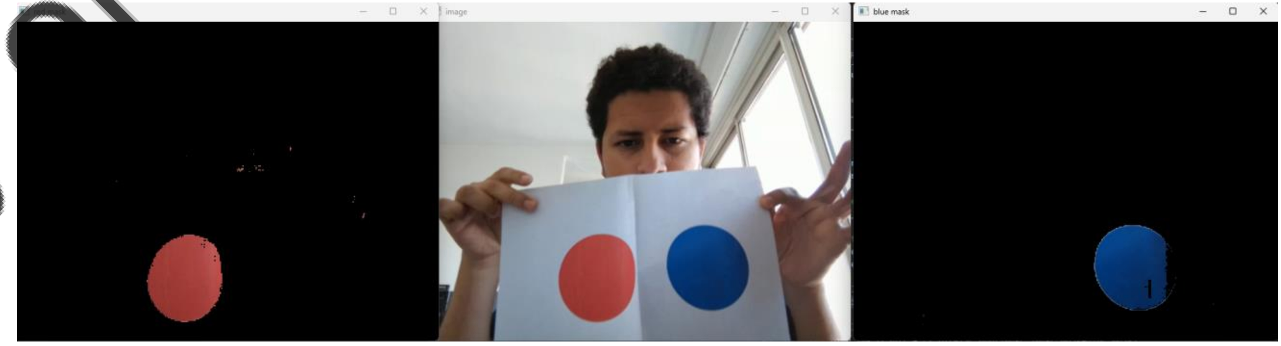
Colour Filtering Algorithm

The Following algorithm will Filter the blue and red colours form the video “sample_video.mp4” (you can use your own).

1. Convert the base image from RGB to the HSV space
2. Filter the image to remove all colours not defined as red or blue, storing the results in two separate images.
   1. Use OpenCV’s `inRange` function
   2. Suggested values for the colours required

	Red	Blue
H	0-33	95-110
S	90-255	80-255
V	130-255	80-255

3. Apply a mask with a bitwise AND operation to view only the colour blue or red.



Colour Filtering

```python
import cv2 as cv
import numpy as np

#HSV Values for Red Colour
H_red_low_limit = 0
S_red_low_limit = 95
V_red_low_limit = 130

H_red_up_limit = 5
S_red_up_limit = 255
V_red_up_limit = 255

#HSV Values for Blue Colour
H_blue_low_limit = 95
S_blue_low_limit = 90
V_blue_low_limit = 90

H_blue_up_limit = 110
S_blue_up_limit = 255
V_blue_up_limit = 255

if __name__=="__main__":
    #cap = cv.VideoCapture(2)
    video_path = 'videos/sample_video.mp4'
    cap = cv.VideoCapture(video_path)

    if not cap.isOpened():
        print("Cannot open camera")
```

```python
    #HSV Values for Green Colour
    while True:
        # Reading frame from video or webcam
        ret, image = cap.read()
        # if frame is read correctly ret is True
        if not ret:
            print("Can't receive frame (stream end?). Exiting ...")
            break
        k = cv.waitKey(25) & 0xFF
        if k == 27:
            break

        # Convert BGR to HSV
        hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

        #Ranges
        red_lower_limit = np.array([H_red_low_limit, S_red_low_limit, V_red_low_limit])
        red_upper_limit = np.array([H_red_up_limit, S_red_up_limit, V_red_up_limit])
        blue_lower_limit = np.array([H_blue_low_limit, S_blue_low_limit, V_blue_low_limit])
        blue_upper_limit = np.array([H_blue_up_limit, S_blue_up_limit, V_blue_up_limit])

        #Masks
        red_mask = cv.inRange(hsv_image, red_lower_limit, red_upper_limit)
        blue_mask = cv.inRange(hsv_image, blue_lower_limit, blue_upper_limit)

        #Partial masks applied to the image (AND Function with the image)
        red_img_result = cv.bitwise_and(image,image,mask=red_mask)
        blue_img_result = cv.bitwise_and(image,image,mask=blue_mask)

        #Full red mask applied to te image
        cv.imshow('image', image)
        cv.imshow('red mask', red_img_result)
        cv.imshow('blue mask', blue_img_result)

    cap.release()
    cv.destroyAllWindows()
```

Noise Rejection

- It is common for noise, such as other objects with the same colour intensity, to appear in the images after applying certain operations.

- One of the most common techniques to deal with these artefacts is known as morphological operators.

Noisy image

After some initial operations to an original image, we can see some noise in the result that we want to remove

Thresholding, Masks and Bitwise Operations

Thresholding

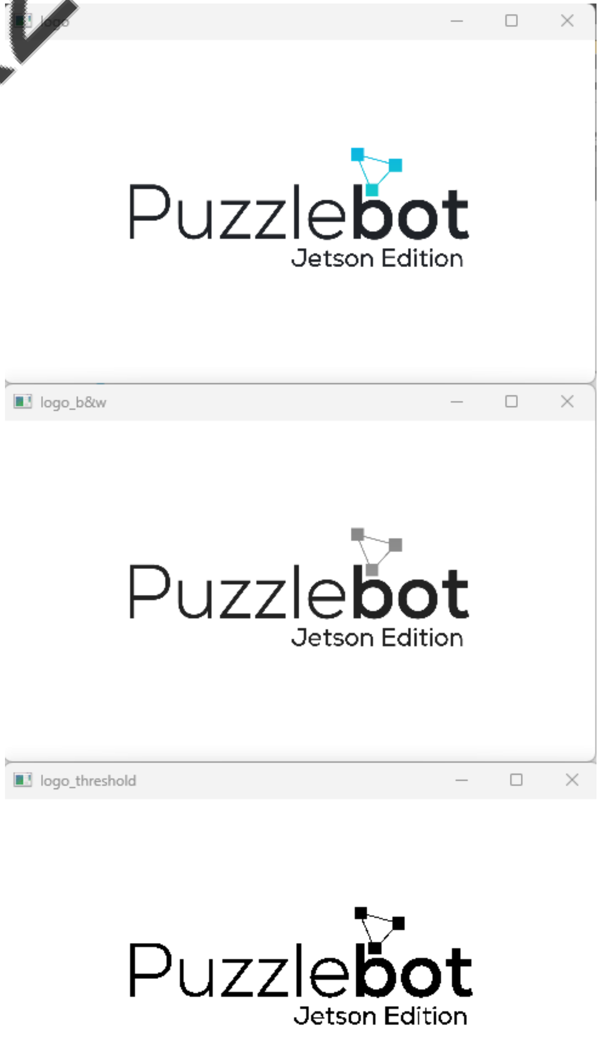
Converts a grayscale image into a binary image. More information [here](#).

Basic Idea:

- If pixel intensity > threshold, set to max value (e.g., 255).
- Else, set to 0.

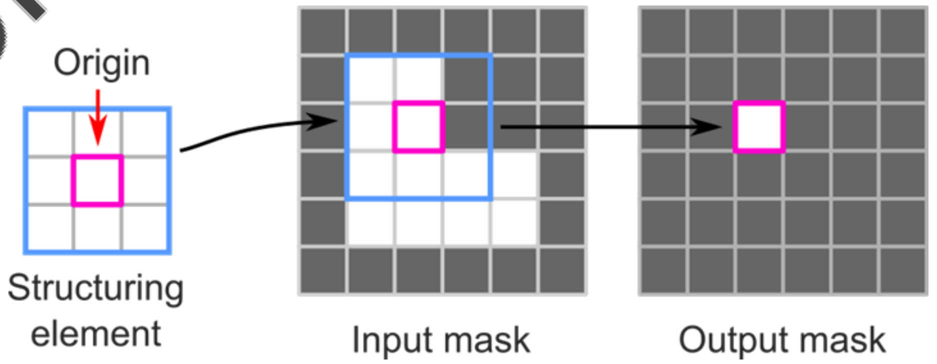
- Function: `cv2.threshold(source, threshold (0-255), max_value, threshold_type)`

```
img_gray = cv.cvtColor(img,cv.COLOR_BGR2GRAY)

ret, mask = cv2.threshold(img_gray, 10, 255, cv2.THRESH_BINARY)

## ret: the threshold used.
## mask: binary image result.
```



Morphological Operators

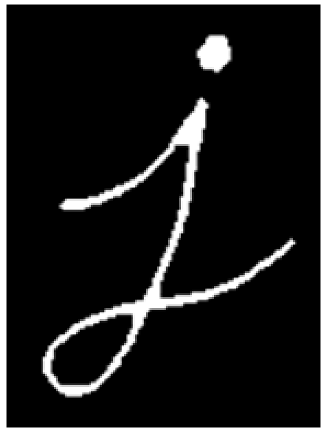
- Morphological operators are defined as a combination between an image and a structuring element.
- The structuring element is usually called "kernel".
- Some operations can be induced by convolving (convolution) the kernel with the original image.
- The most basic morphological operations are Erosion and Dilation.



Morphological Operators

Erosion

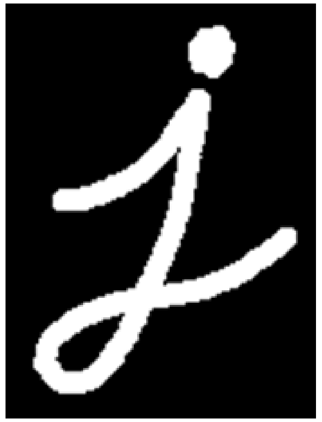
- It computes a local minimum over the area of a given kernel.
- Using a mask (kernel) , we compute the minimal value of a given area around a pixel, and we replace it with that value.
- The results of this technique are shrinking the larger regions and removing the smallest ones.

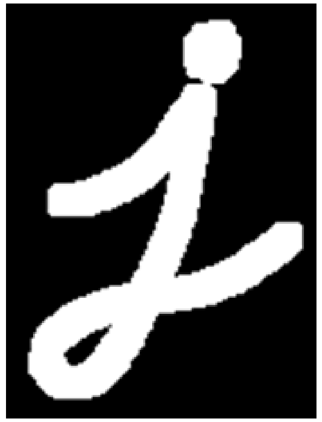


Morphological Operators

Dilation

- Analogous to the Erosion.
- It computes a local maximum over the area of a given kernel.
- Using a mask (kernel), we compute the maximum value of a given area around a pixel, and we replace it with that value.
- The results of this technique are in shapes becoming larger and smoothing their edges.



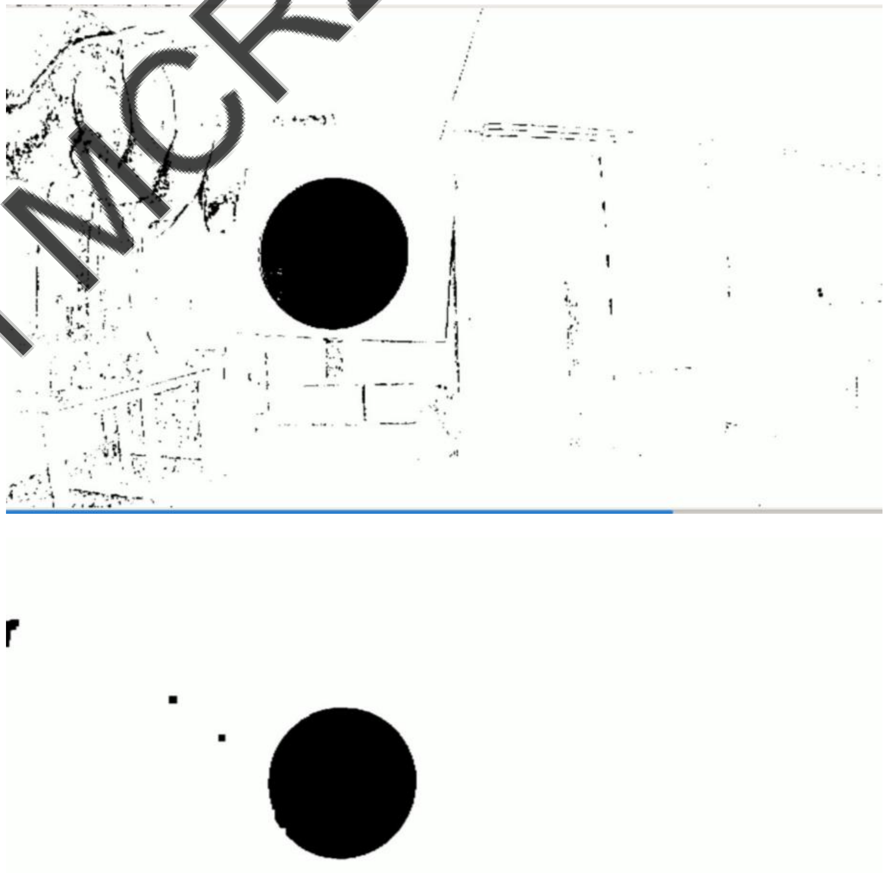


Using morphological operators in OpenCV

To apply any morphological operator, we will need to define 3 things.

1. *The mask/kernel:* it will define the area of effect of our operation, for example in an erosion the bigger the mask the smaller will the final shape be.
2. *Type of operation:* OpenCV has functions defining the most common operators, so it is a matter of selecting the suitable option
3. *Number of iterations:* It is common to apply these operations recursively, so OpenCV allows setting the number of iterations that are going to be made.

More information can be found here.



Activity 2

Morphological Operations

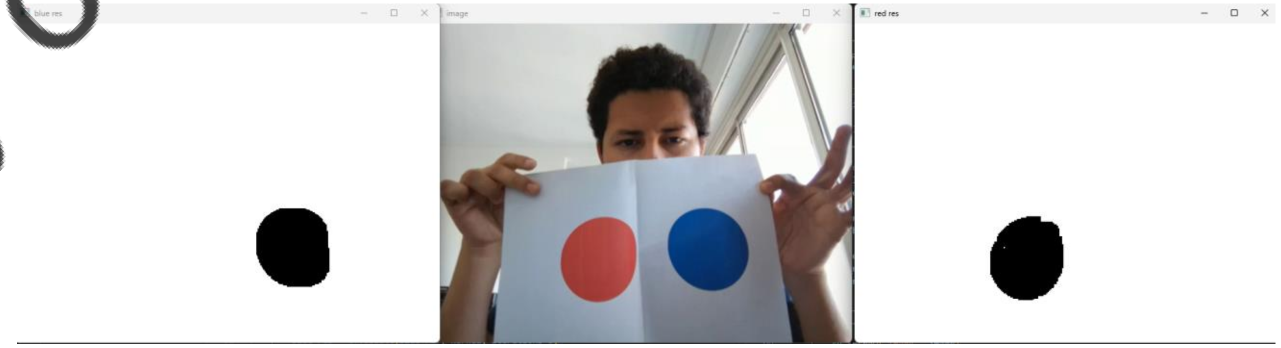
{Learn, Create, Innovate};

MCR2 Manchester Robotics

Activity 2

- Using the code in Activity 3, morphological operations such as erosion and dilation will be implemented.

1. Transform the masked image to grayscale.
2. Threshold the image to remove areas with low intensity.
3. Erode the image to remove the noise.
4. Dilate the image to smooth edges and enlarge the figures.



Colour Filtering

```python
import cv2 as cv
import numpy as np

#HSV Values for Red Colour
H_red_low_limit = 0
S_red_low_limit = 95
V_red_low_limit = 130

H_red_up_limit = 5
S_red_up_limit = 255
V_red_up_limit = 255

#HSV Values for Blue Colour
H_blue_low_limit = 95
S_blue_low_limit = 90
V_blue_low_limit = 90

H_blue_up_limit = 110
S_blue_up_limit = 255
V_blue_up_limit = 255

if __name__=="__main__":
    #cap = cv.VideoCapture(2)
    video_path = 'videos/sample_video.mp4'
    cap = cv.VideoCapture(video_path)

    if not cap.isOpened():
        print("Cannot open camera")
```

```python
#HSV Values for Green Colour
while True:
    # Reading frame from video or webcam
    ret, image = cap.read()
    # if frame is read correctly ret is True
    if not ret:
        print("Can't receive frame (stream end?). Exiting ...")
        break
    k = cv.waitKey(25) & 0xFF
    if k == 27:
        break

    # Convert BGR to HSV
    hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

    #Ranges
    red_lower_limit = np.array([H_red_low_limit, S_red_low_limit, V_red_low_limit])
    red_upper_limit = np.array([H_red_up_limit, S_red_up_limit, V_red_up_limit])
    blue_lower_limit = np.array([H_blue_low_limit, S_blue_low_limit, V_blue_low_limit])
    blue_upper_limit = np.array([H_blue_up_limit, S_blue_up_limit, V_blue_up_limit])

    #Masks
    red_mask = cv.inRange(hsv_image, red_lower_limit, red_upper_limit)
    blue_mask = cv.inRange(hsv_image, blue_lower_limit, blue_upper_limit)

    #Partial masks applied to the image (AND Function with the image)
    red_img_result = cv.bitwise_and(image,image,mask=red_mask)
    blue_img_result = cv.bitwise_and(image,image,mask=blue_mask)
```

Colour Filtering

```
    #Transform to Grayscale
    bw_blue_res = cv.cvtColor(blue_img_result, cv.COLOR_BGR2GRAY)
    bw_red_res = cv.cvtColor(red_img_result, cv.COLOR_BGR2GRAY)

    _ , red_res = cv.threshold(bw_red_res, 30, 255, cv.THRESH_BINARY_INV)
    _ , blue_res = cv.threshold(bw_blue_res, 33, 255, cv.THRESH_BINARY_INV)

    kernel = np.ones((5, 5), np.uint8)
    red_res = cv.dilate(red_res, kernel, iterations=3)
    red_res = cv.erode(red_res, kernel, iterations=3)

    blue_res = cv.dilate(blue_res, kernel, iterations=8)
    blue_res = cv.erode(blue_res, kernel, iterations=8)

    cv.imshow('image', image)
    cv.imshow('red mask', red_img_result)
    cv.imshow('blue mask', blue_img_result)
    cv.imshow('blue res', blue_res)
    cv.imshow('red res', red_res)

# When everything done, release the captureq
cap.release()
cv.destroyAllWindows()
```

OpenCV

Blob Detection

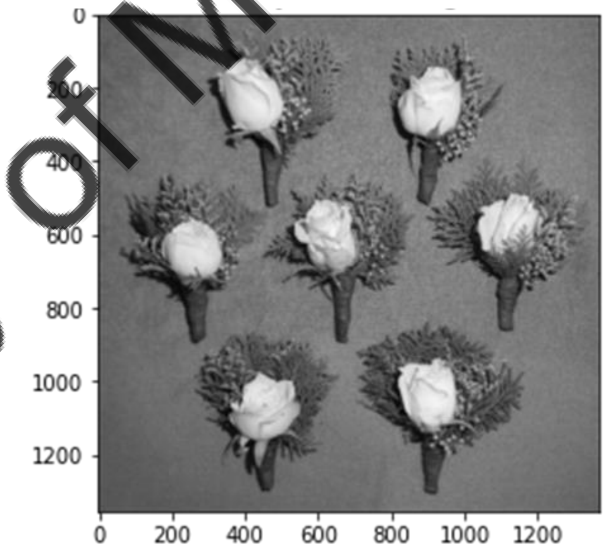
`{Learn, Create, Innovate};`

MCR² Manchester **Robotics**

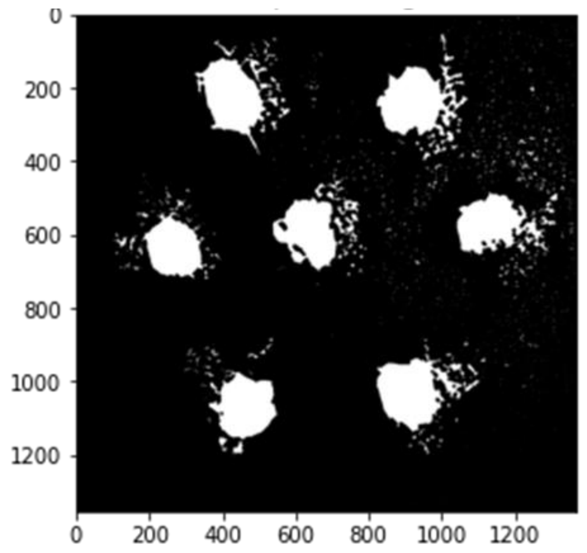
What is blob?

- A Blob is a group of connected pixels in an image that share some common property.
- Usually are areas in an image that are visually distinct, often brighter/darker.
- In the image, the dark connected regions are blobs, and the goal of blob detection is to identify and mark these regions.
- Useful to detect objects (balls, screws, parts).
- Tracking moving elements.
- Obstacle detection.

➡ Blob = "Binary Large Object"



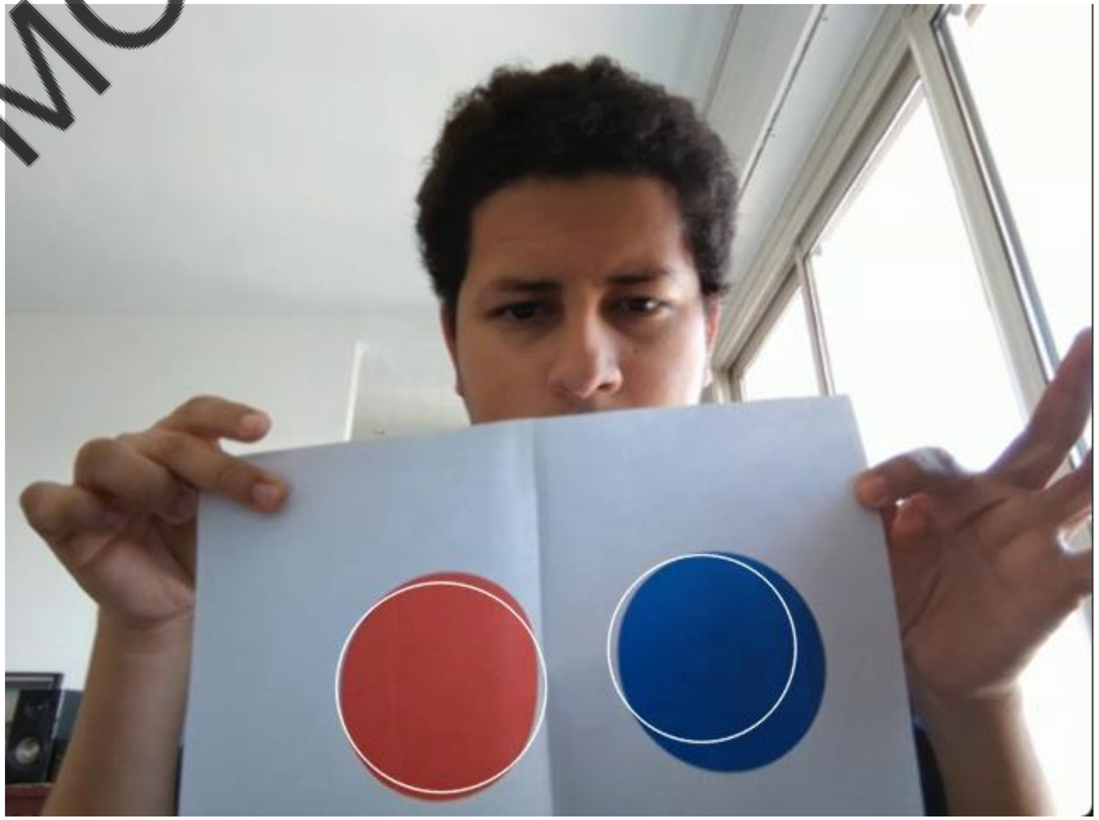
Grayscale image



Resulting blobs after processing

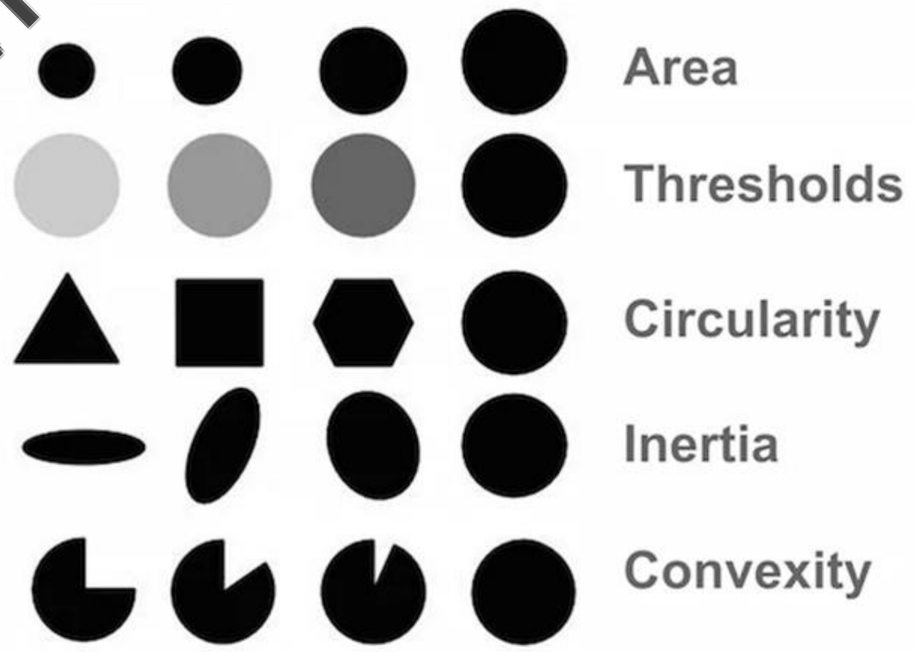
What is blob?

- Blob detection involves five steps:
  - **Thresholding**: convert the input image to a binary image through thresholding.
  - **Grouping**: Find and cluster connected pixels (regions), by making use of the "findContours" function. The centre is identified for each of the extracted groups of pixels and the value for the centres is computed.
  - **Merging:** Merges all nearby clusters.
  - **Radius and centres:** Compute radiuses and centres of the blobs.
  - **Filtering**: Keep blobs matching specific criteria:
    - Size
    - Circularity
    - Convexity
    - Inertia ratio (elongation), etc.



Blob Filtering

- **Color [0,255]:** Compares the intensity of a binary image at the center of a blob to the parameter *blobColor*.
  - blobColor [0,255] to extract dark blobs and light blobs respectively.
- **Area [0, maxArea]:** Extracted blobs have an area between *minArea* and *maxArea* parameters.
- **Circularity [0,1]:** Extracted blobs have circularity between *minCircularity* and *maxCircularity (Circularity of the Circle is 1).*
- **Ratio [0,1]:** Measures the inertia (elongation)of the blob. Extracted blobs have this ratio between *minInertiaRatio* and *maxInertiaRatio.*
- **Convexity [0,1]:** Extracted blobs have convexity (area / area of blob convex hull) between *minConvexity* and *maxConvexity.*



Tuning Parameters

Parameter	What it Does	Typical Use
Threshold	Minimum grayscale intensity for detection.	Lower limit
filterByArea	Only blobs within specific area.	Filter noise
filterByCircularity	Select blobs close to circular shape.	Detect more specific shapes such as circles
filterByConvexity	How convex the blob shape is.	Avoid hollow shapes
filterByInertia	Measures blob elongation.	Exclude thin blobs
filterByColor	Check colour of the blob	Exclude blobs of different colours.

8 SdNf Ydi n

Simple Blob Detection

{Learn, Create, Innovate};

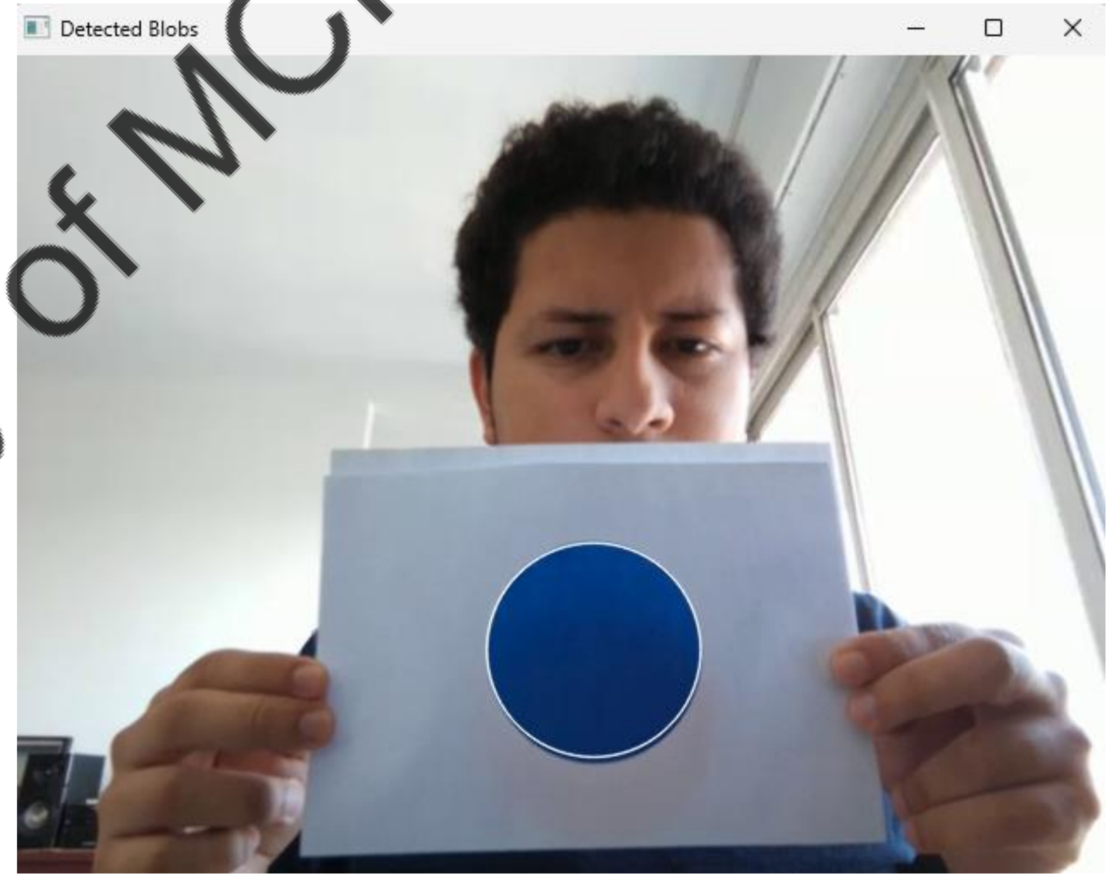
MCR2 Manchester Robotics

Activity 3

Using the code in Activity 4, blob detection will be used to detect the blue blob in a video.

For this activity "`sample_video.mp4`" must be inside the folder "videos"

1. Transform the masked image to grayscale.
2. Threshold the image to remove areas with low intensity.
3. Erode the image to remove the noise.
4. Dilate the image to smooth edges and enlarge the figures.
5. Detect blobs
6. Print centres and blobs.



```python
import cv2 as cv
import numpy as np

# HSV Ranges for detecting BLUE color
H_blue_low_limit = 95
S_blue_low_limit = 90
V_blue_low_limit = 90
H_blue_up_limit = 110
S_blue_up_limit = 255
V_blue_up_limit = 255

# Configuration of OpenCV SimpleBlobDetector Parameters
blob_detec_params = cv.SimpleBlobDetector_Params()

# Thresholds for intensity difference (pixel brightness) during detection
blob_detec_params.minThreshold = 30
blob_detec_params.maxThreshold = 255

# Filter blobs by color (white blobs if 255)
blob_detec_params.filterByColor = True
blob_detec_params.blobColor = 255

# Filter blobs by area (size in pixels)
blob_detec_params.filterByArea = True
blob_detec_params.minArea = 30          # Minimum blob area
blob_detec_params.maxArea = 10000000    # Maximum blob area

# Filter blobs by convexity (how convex the shape is)
blob_detec_params.filterByConvexity = True
blob_detec_params.minConvexity = 0.1
blob_detec_params.maxConvexity = 1

# Filter blobs by circularity (how circular the blob is)
blob_detec_params.filterByCircularity = True
blob_detec_params.minCircularity = 0.5
blob_detec_params.maxCircularity = 1

# Filter blobs by inertia (how elongated the blob is)
blob_detec_params.filterByInertia = True
blob_detec_params.minInertiaRatio = 0.5
blob_detec_params.maxInertiaRatio = 1
```

```python
# Main Code Execution Starts Here
if __name__ == "__main__":

    # Start video capture
    # You can switch to a webcam by using cv.VideoCapture(0)
    video_path = 'videos/sample_video.mp4'
    cap = cv.VideoCapture(video_path)

    if not cap.isOpened():
        print("Cannot open camera or video file")

    # Create the blob detector object with configured parameters
    detector = cv.SimpleBlobDetector_create(blob_detec_params)

    # Main loop to process each frame
    while True:

        # Capture frame from video
        ret, image = cap.read()

        # Exit the loop if no frames are returned (end of video)
        if not ret:
            print("Can't receive frame (stream end?). Exiting ...")
            break

        # Check for ESC key press to exit early
        k = cv.waitKey(25) & 0xFF
        if k == 27:
            break
```

```python
# Step 1: Convert image from BGR to HSV color space
hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

        # Step 2: Create masks for red and blue colors
# Blue mask: a single range
blue_lower_limit = np.array([H_blue_low_limit,
S_blue_low_limit, V_blue_low_limit])
blue_upper_limit = np.array([H_blue_up_limit, S_blue_up_limit,
V_blue_up_limit])

        # Apply color thresholds to create binary masks
blue_mask = cv.inRange(hsv_image, blue_lower_limit,
blue_upper_limit)

        # Step 3: Apply masks to the original image for
visualization
# Apply blue mask
blue_img_result = cv.bitwise_and(image, image, mask=blue_mask)

        # Step 4: Convert the masked images to grayscale for
further processing
bw_blue_res = cv.cvtColor(blue_img_result, cv.COLOR_BGR2GRAY)

        # Step 5: Apply binary thresholding to highlight blobs
in binary images
_, blue_res = cv.threshold(bw_blue_res, 33, 255,
cv.THRESH_BINARY)
```

```python
    # Step 6: Perform morphological operations to clean up noise
    # Create a kernel for morphological operations
    kernel = np.ones((5, 5), np.uint8)

            # Clean blue mask: more aggressive clean-up
    blue_res = cv.erode(blue_res, kernel, iterations=8)
    blue_res = cv.dilate(blue_res, kernel, iterations=8)

            # Step 7: Detect blobs in the cleaned binary images
    bluekeypoints = detector.detect(blue_res)

            # Step 9: Draw keypoints (blobs) on the original image
    view_blobs = cv.drawKeypoints(image, bluekeypoints,
    np.array([]),
                                  (255, 255, 255),
    cv.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

            # Step 10: Display the images for
    visualization/debugging
    #cv.imshow('Original Image', image)
    #cv.imshow('Blue Mask Applied', blue_img_result)
    #cv.imshow('Blue Binary Mask', blue_res)
    cv.imshow('Detected Blobs', view_blobs)

        # Step 11: Release resources and close windows when done
cap.release()
cv.destroyAllWindows()
```

OpenCV

OpenCV in ROS

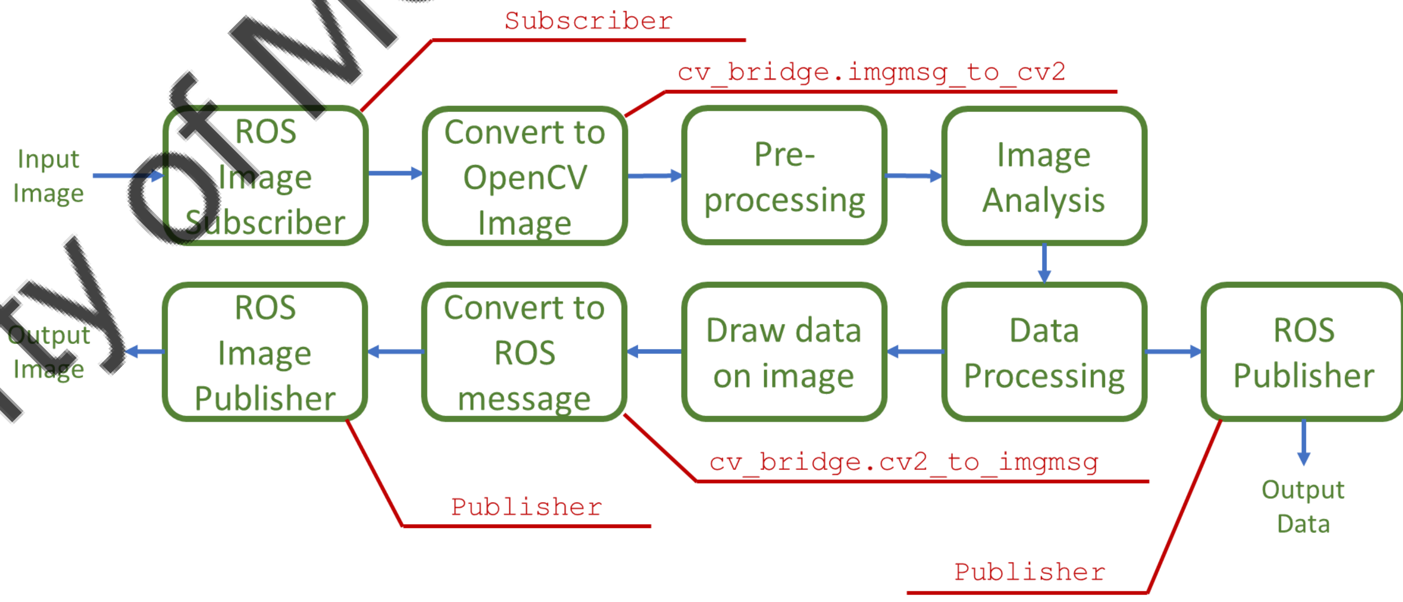
MCR2 Manchester Robotics

Interfacing with ROS

- ROS uses a package called CV Bridge to interface with OpenCV:

```
$ sudo apt install ros-humble-cv-bridge
```

- This package, converts between ROS Image messages and OpenCV images.
- More information can be found here.
- OpenCV and cv_bridge come preinstalled on the Jetson.

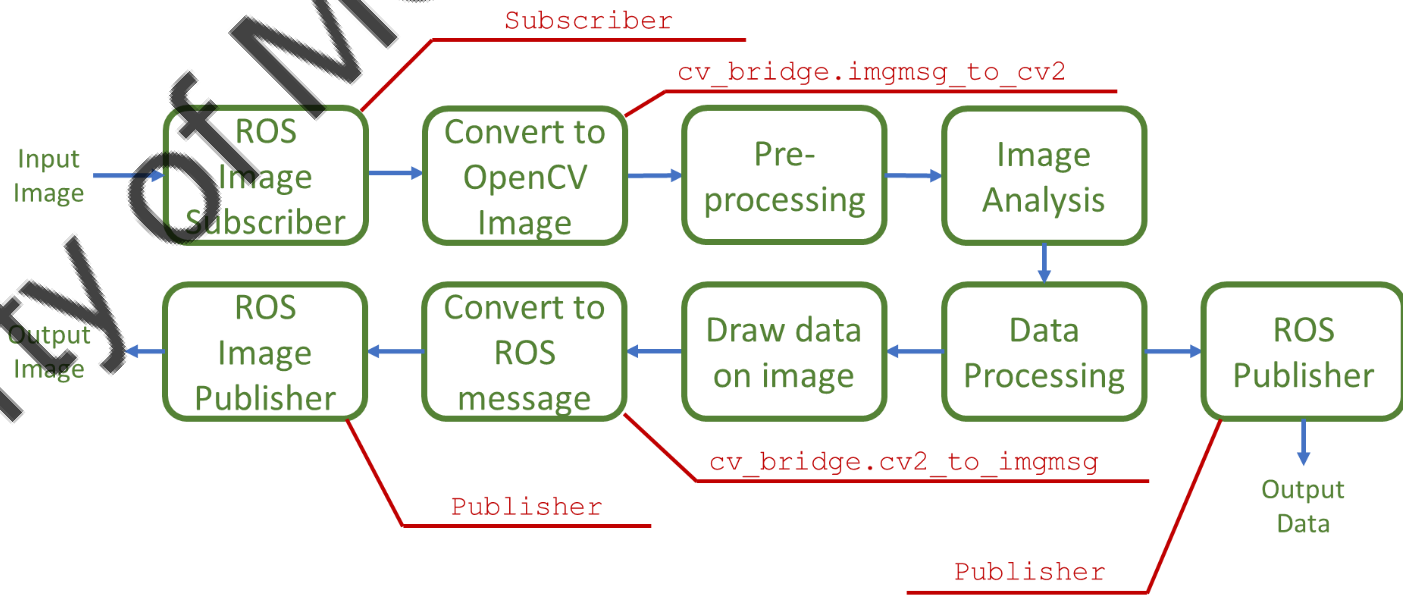


Interfacing with ROS

- ROS uses a package called CV Bridge to interface with OpenCV:

```
$ sudo apt install ros-humble-cv-bridge
```

- This package, converts between ROS Image messages and OpenCV images.
- More information can be found here.
- OpenCV and cv_bridge come preinstalled on the Jetson.



Activity

E` U^9L $ HEI

{Learn, Create, Innovate};

Manchester **Robotics**

OpenCV and ROS

Objective

- The objective of this activity is to show the user how ROS and OpenCV can be used together.
- In this activity, some simple feature extraction techniques will be used.

Requirements

- The following "usb_cam" node must be installed by the user.

```
$ sudo apt install ros-humble-usb-cam
```

- The node can be run using the following command

```
$ ros2 run usb_cam usb_cam_node_exe
```

- This node will access the camera and publish the image into the topic "/image_raw"

Puzzlebot (not necessary)

- If using the Puzzlebot, the user can follow the instruction manual or the presentation "MCR2_Puzzlebot_Jetson_Ed" to activate the camera using the command

```
$ ros2 launch ros_deep_learning video_source.ros2.launch
```

- The same node can be easily launched from a (launch file) using the following command

```
$ ros2 launch puzzlebot_ros camera_jetson.launch.py
```

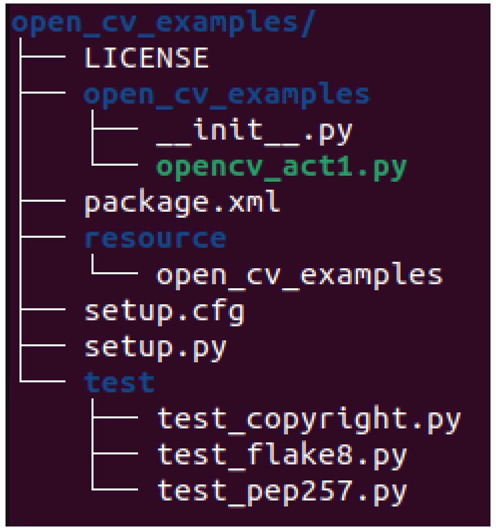
- This node will access the camera and publish the image into the topic "/video_source/raw"

OpenCV and ROS

Instructions

- Create a new package

```
ros2 pkg create --build-type ament_python open_cv_examples
--license Apache-2.0 --dependencies rclpy sensor_msgs
opencv-python cv_bridge ros2launch std_msgs python3-numpy
--node-name opencv_act1 --maintainer-name 'Mario Martinez'
--maintainer-email 'mario.mtz@manchester-robotics.com' --
description 'OpenCV Examples'
```



```
open_cv_examples/
├── LICENSE
├── open_cv_examples
│   ├── __init__.py
│   └── opencv_act1.py
├── package.xml
├── resource
│   └── open_cv_examples
├── setup.cfg
├── setup.py
└── test
    ├── test_copyright.py
    ├── test_flake8.py
    └── test_pep257.py
```

- Give executable permission to the file

```
$ cd ~/src/open_cv_examples/open_cv_examples/
$ sudo chmod +x opencv_act1.py
```

- Edit the *opencv_act1.py* file.

```python
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
import cv2 as cv
from cv_bridge import CvBridge
import numpy as np

class ColorDetectionNode(Node):

    def __init__(self):
        super().__init__('color_detection_node')

        # Initialize variables
        self.img = None
        self.bridge = CvBridge()

        # Define HSV range for blue color detection
        self.blue_lower = np.array([110, 60, 60], np.uint8)
        self.blue_upper = np.array([160, 255, 255], np.uint8)

        # ROS2 Subscriber for raw camera images
        self.subscription = self.create_subscription(
            Image,
            '/image_raw',
            self.camera_callback,
            10
        )

        # ROS2 Publisher for processed images
        self.image_pub = self.create_publisher(
            Image,
            '/image_processing/image',
            10
        )

        # Timer setup for regular processing
        self.timer = self.create_timer(0.1, self.timer_callback)

        self.get_logger().info('Color Detection Node has started!')
```

```python
    def camera_callback(self, msg):
        """Callback to receive image from the camera and convert to CV2
format."""
        try:
            self.img = self.bridge.imgmsg_to_cv2(msg, "bgr8")
        except Exception as e:
            self.get_logger().error(f'Conversion error: {e}')

    def timer_callback(self):
        """Timer callback for processing the image."""

        # Check if image has been received
        if self.img is None:
            self.get_logger().info('Waiting for image data...')
            return

        # Step 1: Reduce noise with Gaussian blur
        blurred_img = cv.GaussianBlur(self.img, (9, 9), 2)
        # Step 2: Convert from BGR to HSV color space
        hsv_img = cv.cvtColor(blurred_img, cv.COLOR_BGR2HSV)
        # Step 3: Create binary mask for blue color
        blue_mask = cv.inRange(hsv_img, self.blue_lower, self.blue_upper)
        # Step 4: Apply mask to extract blue regions from original image
        blue_extracted_img = cv.bitwise_and(self.img, self.img, mask=blue_mask)
        # Step 5: Convert extracted blue regions to grayscale
        gray_blue_img = cv.cvtColor(blue_extracted_img, cv.COLOR_BGR2GRAY)
        # Step 6: Threshold grayscale image to binary image
        _, binary_blue_img = cv.threshold(gray_blue_img, 5, 255,
cv.THRESH_BINARY)

        # Step 7: Apply morphological operations to clean noise
        kernel = np.ones((3, 3), np.uint8)
        cleaned_img = cv.erode(binary_blue_img, kernel, iterations=8)
        cleaned_img = cv.dilate(cleaned_img, kernel, iterations=8)

        # Publish the cleaned binary image
        processed_img_msg = self.bridge.cv2_to_imgmsg(cleaned_img,
encoding='mono8')
        self.image_pub.publish(processed_img_msg)
```

OpenCV and ROS

```
def main(args=None):
    rclpy.init(args=args)
    color_detection_node = ColorDetectionNode()

    try:
        rclpy.spin(color_detection_node)
    except KeyboardInterrupt:
        pass
    finally:
        color_detection_node.destroy_node()
        if rclpy.ok():
            rclpy.shutdown()


if __name__ == '__main__':
    main()
```

- In a new terminal build the new package, source it, and run it

```
cd ~/ros2_ws/
colcon build
source install/setup.bash
```

- Start the camera

```
$ ros2 run usb_cam usb_cam_node_exe
```

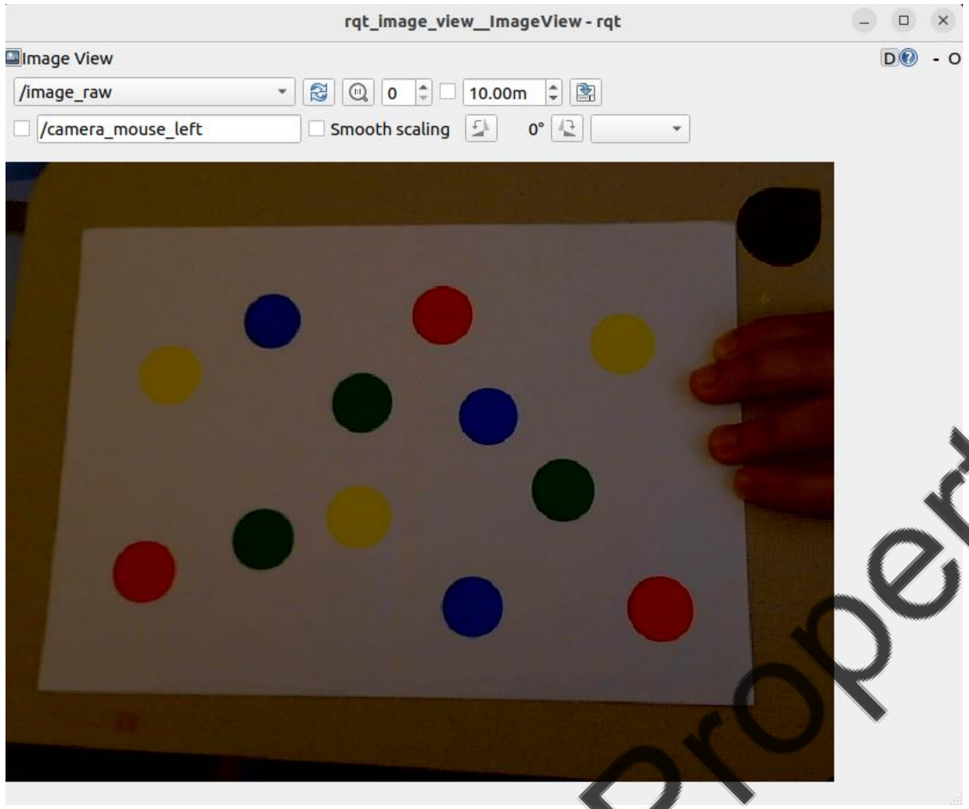
- Run the node

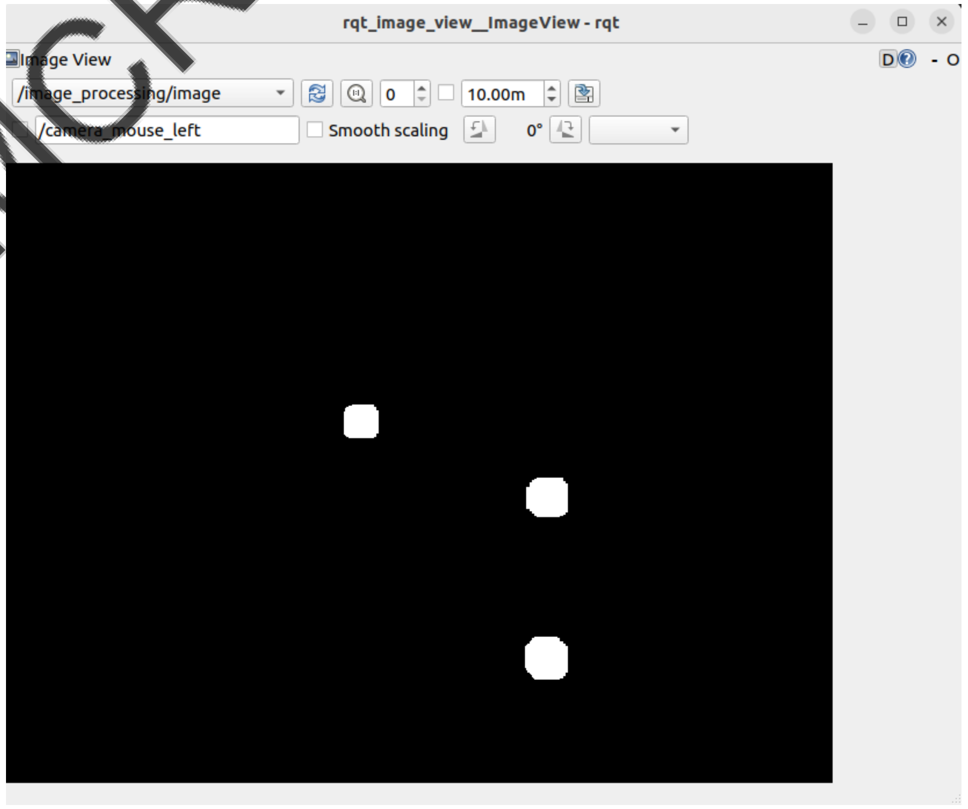
```
$ ros2 run open_cv_examples opencv_act1
```

- Open RVIZ, or rqt_image_view to view the results

```
$ ros2 run rqt_image_view rqt_image_view
```

OpenCV and ROS





8 SaNf Yai

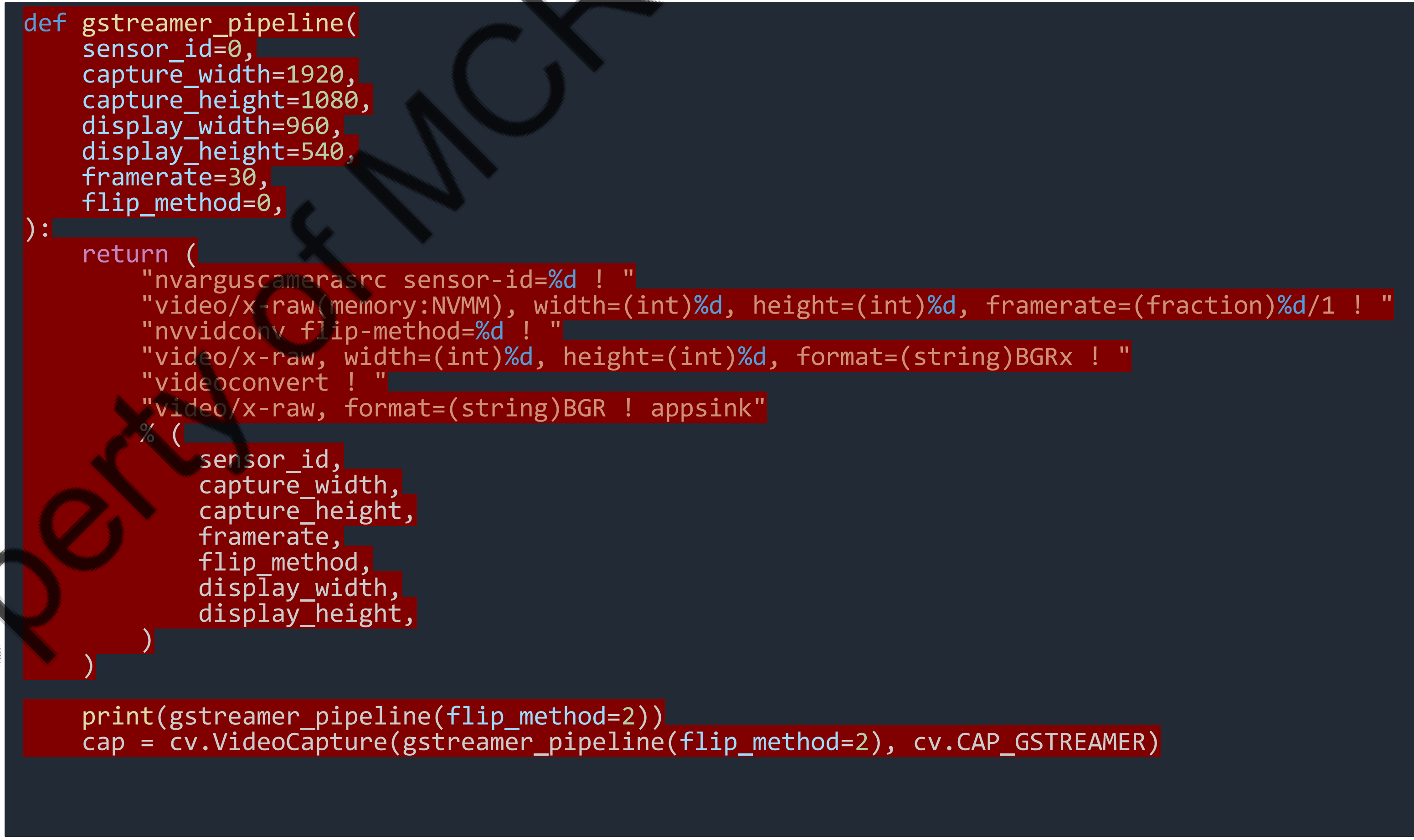
OpenCV in Puzzlebot
- No ROS

{Learn, Create, Innovate};

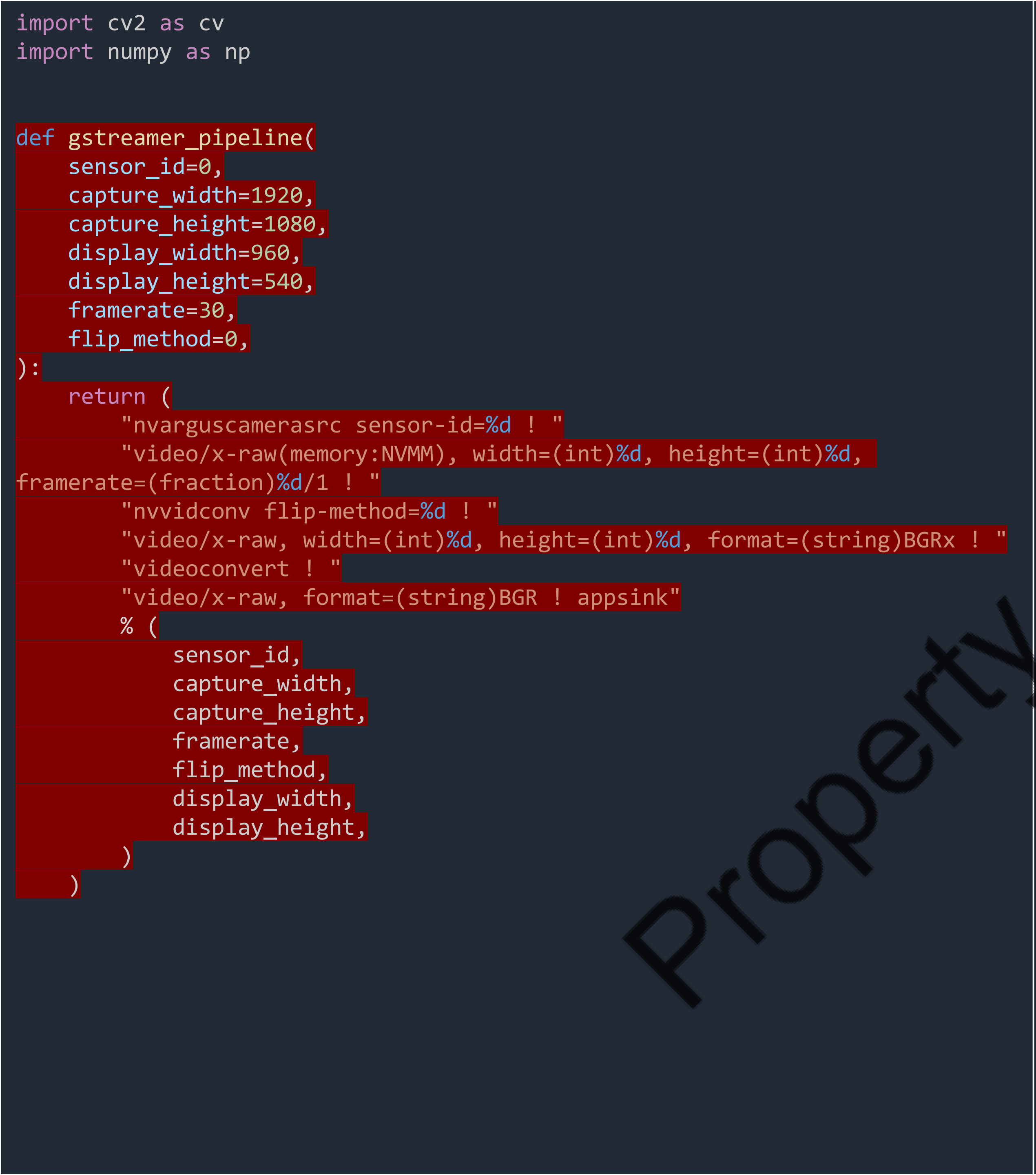
MCR2
Manchester **Robotics**

OpenCV in Puzzlebot – No ROS

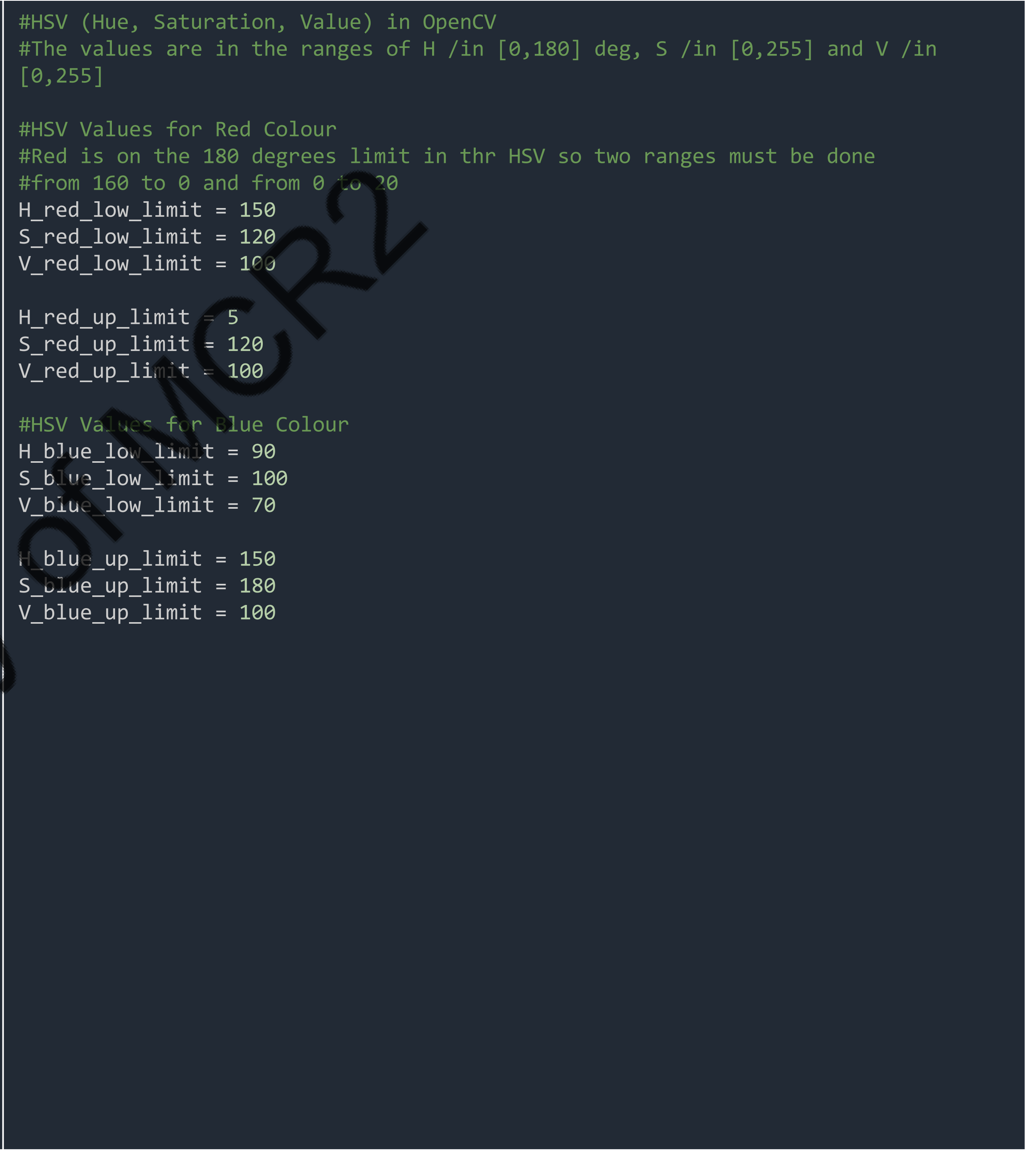
- In the Puzzlebot, the access to the camera used in OpenCV differs from that of a regular camera.
- The Puzzlebot contains an IMX19 Camera module (Raspberry Pi camera) to access such a camera, the user must declare it at the top of the OpenCV Script as stated in the right image.
- The following code shows a simple masking activity on the Puzzlebot. The activity shows a correct way to activate the camera on the Puzzlebot and use it in an OpenCV script.



```
def gstreamer_pipeline(
    sensor_id=0,
    capture_width=1920,
    capture_height=1080,
    display_width=960,
    display_height=540,
    framerate=30,
    flip_method=0,
):
    return (
        "nvarguscamerasrc sensor-id=%d ! "
        "video/x-raw(memory:NVMM), width=(int)%d, height=(int)%d, framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            sensor_id,
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )

    print(gstreamer_pipeline(flip_method=2))
    cap = cv.VideoCapture(gstreamer_pipeline(flip_method=2), cv.CAP_GSTREAMER)
```



```python
import cv2 as cv
import numpy as np


def gstreamer_pipeline(
    sensor_id=0,
    capture_width=1920,
    capture_height=1080,
    display_width=960,
    display_height=540,
    framerate=30,
    flip_method=0,
):
    return (
        "nvarguscamerasrc sensor-id=%d ! "
        "video/x-raw(memory:NVMM), width=(int)%d, height=(int)%d, 
framerate=(fraction)%d/1 ! "
        "nvvidconv flip-method=%d ! "
        "video/x-raw, width=(int)%d, height=(int)%d, format=(string)BGRx ! "
        "videoconvert ! "
        "video/x-raw, format=(string)BGR ! appsink"
        % (
            sensor_id,
            capture_width,
            capture_height,
            framerate,
            flip_method,
            display_width,
            display_height,
        )
    )
```

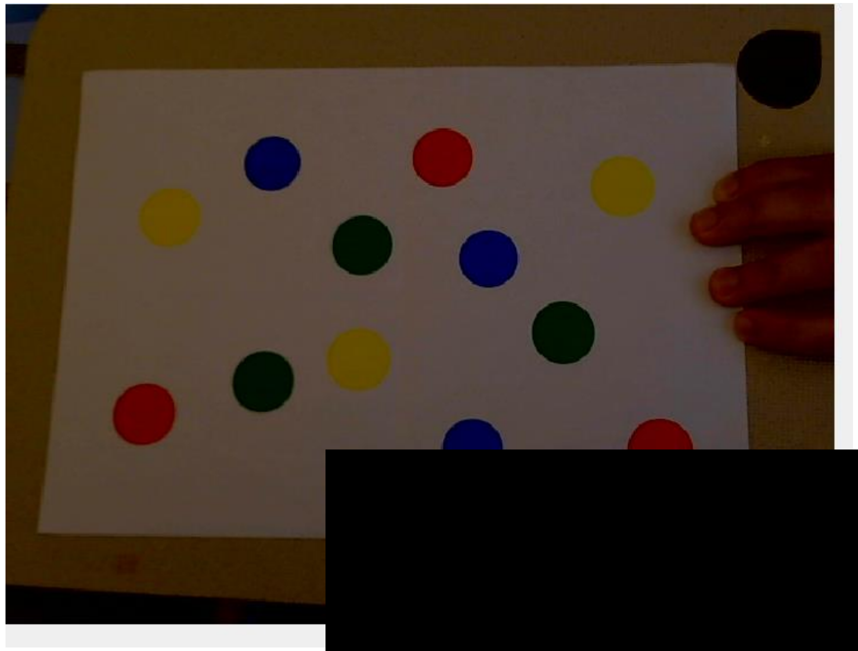


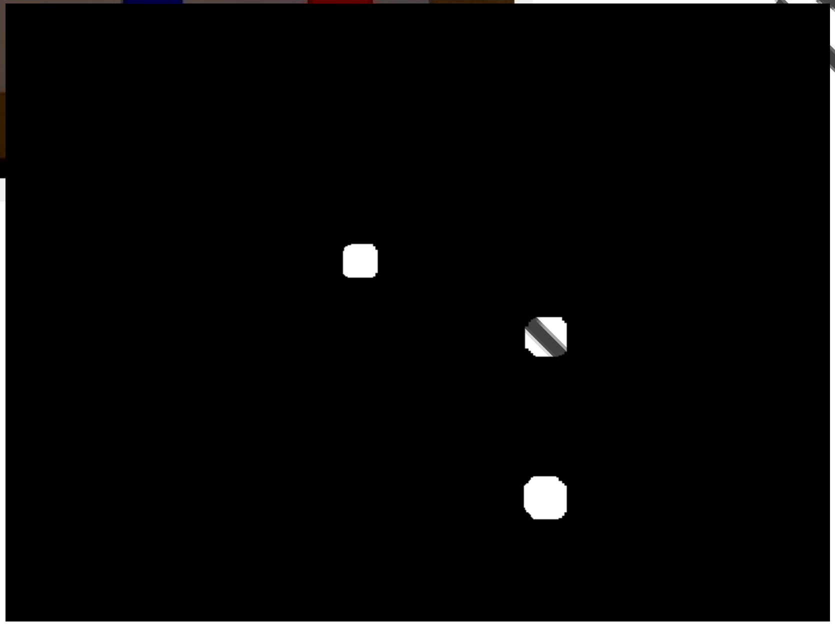
```python
#HSV (Hue, Saturation, Value) in OpenCV
#The values are in the ranges of H /in [0,180] deg, S /in [0,255] and V /in 
[0,255]

#HSV Values for Red Colour
#Red is on the 180 degrees limit in thr HSV so two ranges must be done
#from 160 to 0 and from 0 to 20
H_red_low_limit = 150
S_red_low_limit = 120
V_red_low_limit = 100

H_red_up_limit = 5
S_red_up_limit = 120
V_red_up_limit = 100

#HSV Values for Blue Colour
H_blue_low_limit = 90
S_blue_low_limit = 100
V_blue_low_limit = 70

H_blue_up_limit = 150
S_blue_up_limit = 180
V_blue_up_limit = 100
```

```python
if __name__=="__main__":

    print(gstreamer_pipeline(flip_method=2))
    cap = cv.VideoCapture(gstreamer_pipeline(flip_method=2), cv.CAP_GSTREAMER)
    #window_handle = cv.namedWindow('CSI Camera', cv.WINDOW_AUTOSIZE)

    if cap.isOpened():
        try:
            #HSV Values for Green Colour
            while True:

                # Reading frame from video or webcam
                # and resizing to match when combining them
                ret, image = cap.read()

                # if frame is read correctly ret is True
                if not ret:
                    print("Can't receive frame (stream end?). Exiting ...")
                    break

                k = cv.waitKey(25) & 0xFF
                if k == 27:
                    break

                # Convert BGR to HSV
                hsv_image = cv.cvtColor(image, cv.COLOR_BGR2HSV)

                #Red Ranges
                #Red colour requires 2 ranges because its on the limits of the angles [min_H, 179] to
[0, max_H]
                red_lower_limit1 = np.array([H_red_low_limit, S_red_low_limit, V_red_low_limit])
                red_upper_limit1 = np.array([179, 255, 255])

                red_lower_limit2 = np.array([0, S_red_up_limit, V_red_up_limit])
                red_upper_limit2 = np.array([H_red_up_limit, 255, 255])

                #Blue Ranges
                blue_lower_limit = np.array([H_blue_low_limit, S_blue_low_limit, V_blue_low_limit])
                blue_upper_limit = np.array([H_blue_up_limit, S_blue_up_limit, V_blue_up_limit])

                #Red Masks
                red_mask1 = cv.inRange(hsv_image, red_lower_limit1, red_upper_limit1)
                red_mask2 = cv.inRange(hsv_image, red_lower_limit2, red_upper_limit2)
```

```python
                #Blue Mask
                blue_mask = cv.inRange(hsv_image, blue_lower_limit,
blue_upper_limit)

                #Partial masks applied to the image (AND Function with the
image)
                masked_red1 = cv.bitwise_and(image,image,mask=red_mask1)
                masked_red2 = cv.bitwise_and(image,image,mask=red_mask2)

                #Full red mask applied to te image
                red_img_result = cv.bitwise_or(masked_red1, masked_red2)

                #Full blue mask applied to te image
                blue_img_result = cv.bitwise_and(image,image,mask=blue_mask)

                #Transform to Grayscale
                bw_blue_res = cv.cvtColor(blue_img_result, cv.COLOR_BGR2GRAY)
                bw_red_res = cv.cvtColor(red_img_result, cv.COLOR_BGR2GRAY)

                _ , red_res = cv.threshold(bw_red_res, 20, 255,
cv.THRESH_BINARY)
                _ , blue_res = cv.threshold(bw_blue_res, 20, 255,
cv.THRESH_BINARY)

                kernel = np.ones((3, 3), np.uint8)
                red_res = cv.erode(red_res, kernel, iterations=1)
                red_res = cv.dilate(red_res, kernel, iterations=3)

                blue_res = cv.erode(blue_res, kernel, iterations=1)
                blue_res = cv.dilate(blue_res, kernel, iterations=3)

                cv.imshow('image', image)
                cv.imshow('red mask', red_img_result)
                cv.imshow('blue mask', blue_img_result)
                cv.imshow('blue res', blue_res)
                cv.imshow('red res', red_res)
        finally:
            # When everything done, release the captureq
            cap.release()
            cv.destroyAllWindows()

    else:
        print("Error: Unable to open camera")
```

OpenCV and ROS





- Run the previous code form a terminal.
- Go to your "opencv_ws" (if applicable)
- In a new terminal build the new package, source it, and run it

```
$ python3 activity.py
```

Thank you

{Learn, Create, Innovate};

MCR² Manchester Robotics

T&C

Terms and conditions

{Learn, Create, Innovate};

MCR2 Manchester Robotics

Terms and conditions

- *THE PIECES, IMAGES, VIDEOS, DOCUMENTATION, ETC. SHOWN HERE ARE FOR INFORMATIVE PURPOSES ONLY. THE DESIGN IS PROPRIETARY AND CONFIDENTIAL TO MANCHESTER ROBOTICS LTD. (MCR2). THE INFORMATION, CODE, SIMULATORS, DRAWINGS, VIDEOS PRESENTATIONS ETC. CONTAINED IN THIS PRESENTATION IS THE SOLE PROPERTY OF MANCHESTER ROBOTICS LTD. ANY REPRODUCTION, RESELL, REDISTRIBUTION OR USAGE IN PART OR AS A WHOLE WITHOUT THE WRITTEN PERMISSION OF MANCHESTER ROBOTICS LTD. IS STRICTLY PROHIBITED.*
- *THIS PRESENTATION MAY CONTAIN LINKS TO OTHER WEBSITES OR CONTENT BELONGING TO OR ORIGINATING FROM THIRD PARTIES OR LINKS TO WEBSITES AND FEATURES IN BANNERS OR OTHER ADVERTISING. SUCH EXTERNAL LINKS ARE NOT INVESTIGATED, MONITORED, OR CHECKED FOR ACCURACY, ADEQUACY, VALIDITY, RELIABILITY, AVAILABILITY OR COMPLETENESS BY US.*
- *WE DO NOT WARRANT, ENDORSE, GUARANTEE, OR ASSUME RESPONSIBILITY FOR THE ACCURACY OR RELIABILITY OF ANY INFORMATION OFFERED BY THIRD-PARTY WEBSITES LINKED THROUGH THE SITE OR ANY WEBSITE OR FEATURE LINKED IN ANY BANNER OR OTHER ADVERTISING.*