{Learn, Create, Innovate};

Computer Vision in ROS2

OpenCV Interface

Traffic Light Detection

MCR^2

Manchester Robotics

Activity 1

- Implement a ROS node that computes the robot location using the encoder data
  - It should subscribe to `/wl` and `/wr`, and publish the data to a suitable set of topics
  - The published messages could be a Pose2D message

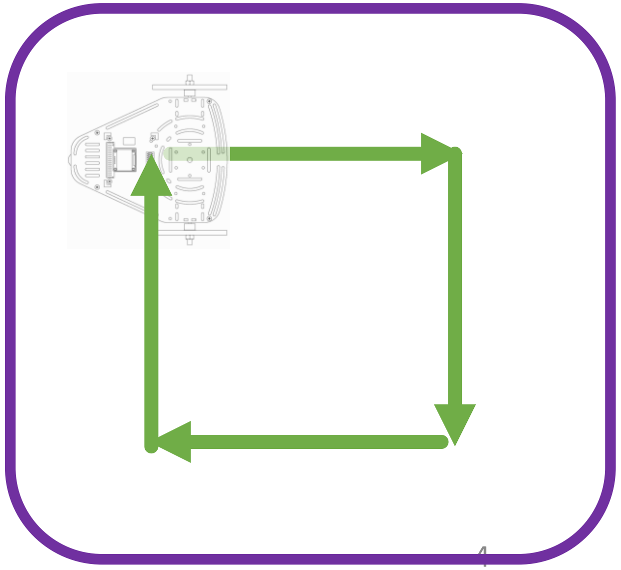
Activity 2

- Modify the previous node to publish e_d and e_θ.
- Set a target, and drive the robot around, checking that the angle to the target and the distance from the target are updated correctly
- Remember to wrap all angles to within 1 circle

Mini challenge

- Use a controller to move the robot to different positions
- The robot must follow a set of consecutive equilateral figures: triangle, square, pentagon, hexagon, …
- Each figure must be contained inside a 1m diameter circle
- The initial pose of the robot must be

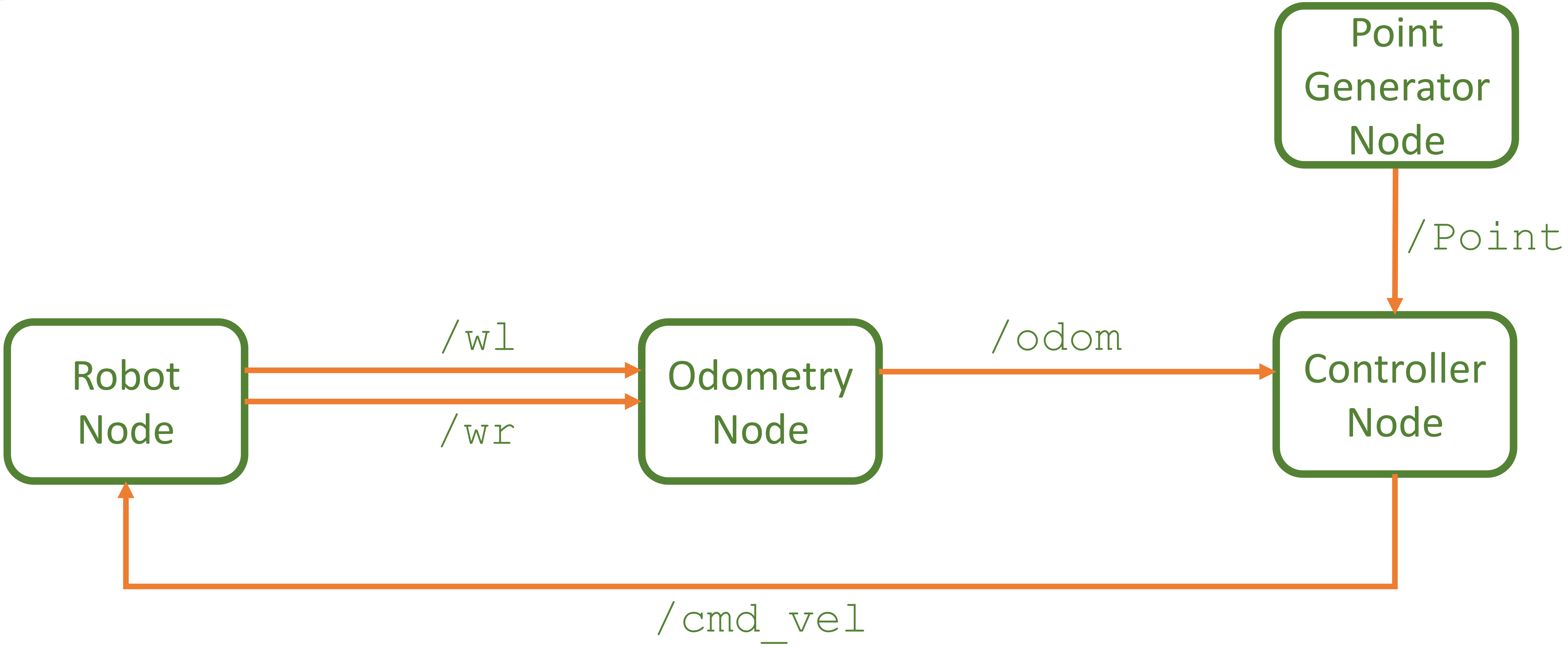
$$[x, y, \theta]^T = [0,0,0]^T$$



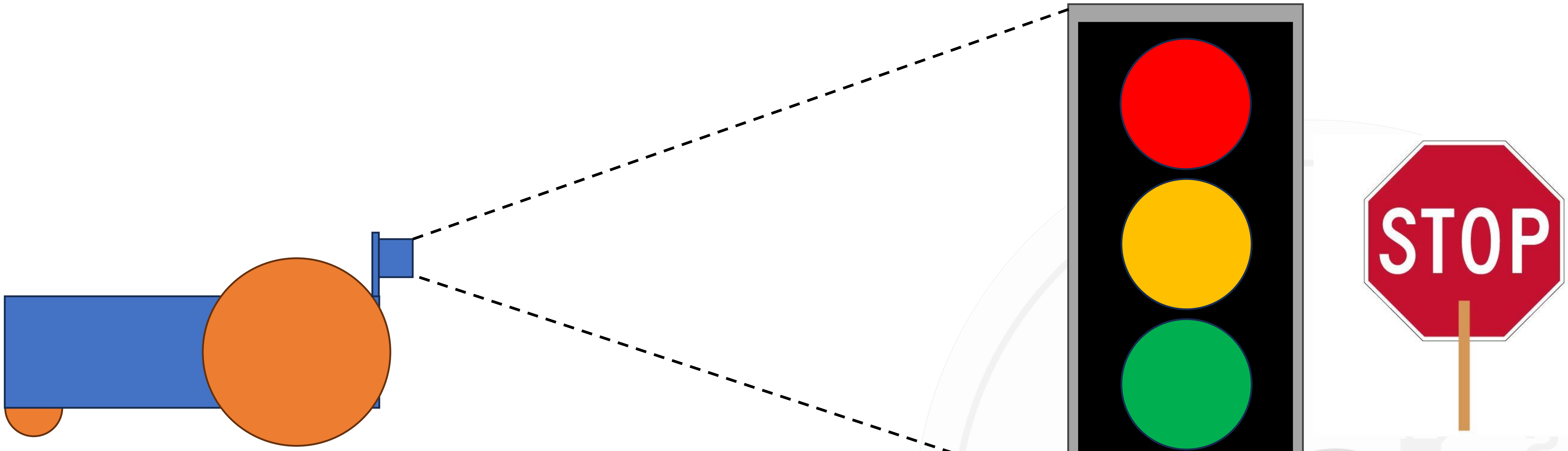
Mini challenge 1

- The open loop controller must be **robust.**
  - The student must define what is robustness and implement strategies to achieve it with the controller.
- The controller must be tunned using a valid methodology
- The controller must take into consideration, perturbation, nonlinearities and noise.
- It is encouraged, but not required, to use a config file and/or parameters to configure the PID, the starting, and finishing figures.

Mini challenge



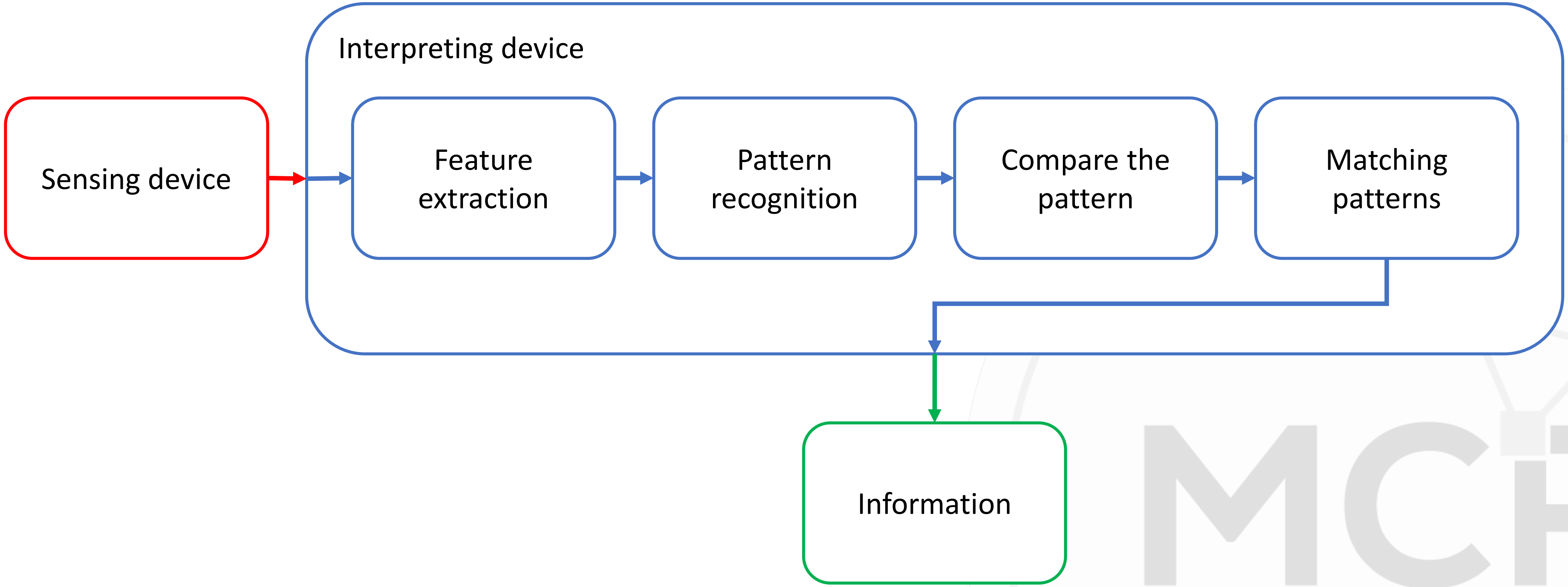
The Task



Computer Vision

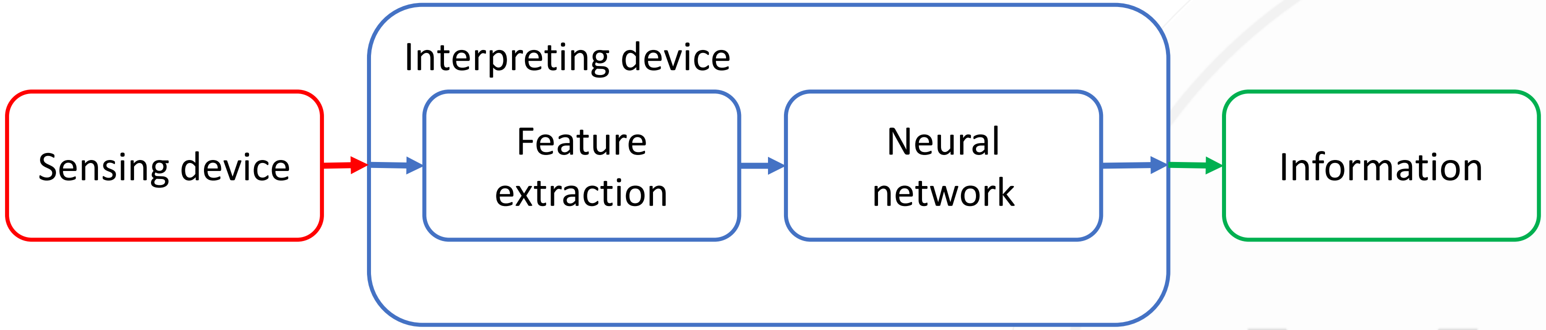
- A sub field in computer science and artificial intelligence dedicated to identify and understand features in an image or video.
- Artificial intelligence mimics the thinking process. Computer vision aims to reproduce human sight and inference.
- Many modern learning-based techniques use computer vision 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 and a convolutional neural network (CNN)

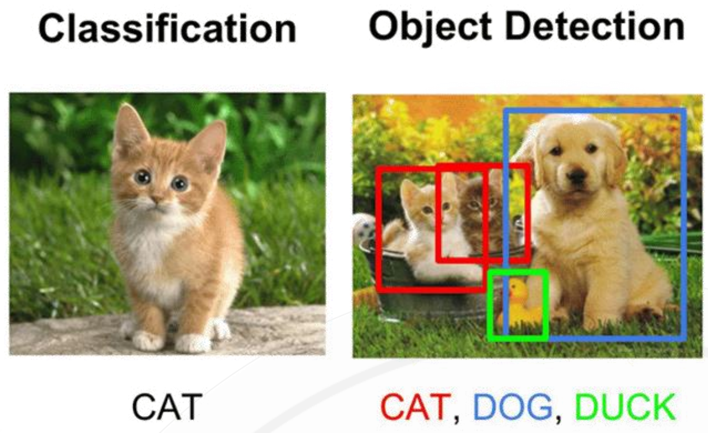


How does learning-based computer vision work?

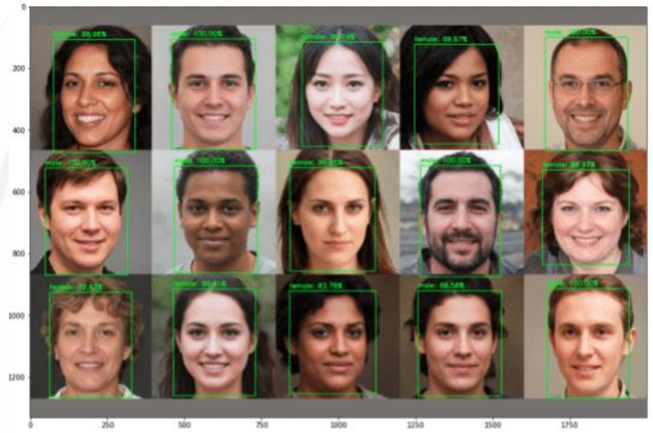
- Deep learning (DL) requires large amounts of data for it to learn about the context of visual data.
  - After several iterations, the model “learns” to differentiate image features.
- CNN decomposes the image into pixels that are labeled.
  - It uses convolutions to predict the tag according to its input pixel.
  - The accuracy of the prediction should increase with time.

Computer vision capabilities

- Object classification
  - Detections are clustered into a category.



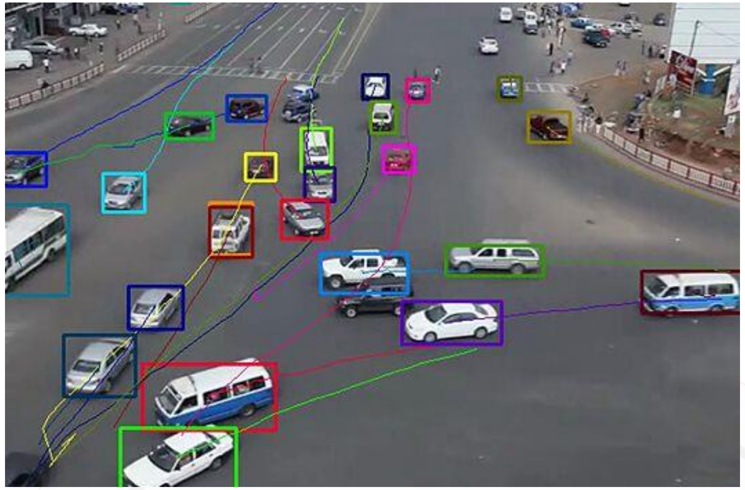
- Object identification
  - Characteristics of the detection are identified.



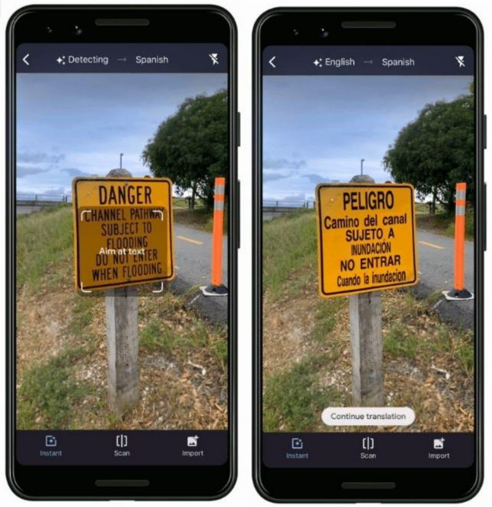
Manchester Robotics

Computer vision capabilities

- Object tracking
  - Process a location of an object over time.



- Optical character recognition
  - Letters and numbers identification to convert it into a set of machine-encoded text



Typical applications

- Content organization
- Agriculture
- Sports
- Face recognition
- Spatial analysis
- Text extraction
- Autonomous vhicles
- Manufacturing
- Augmented reality
- Healthcare

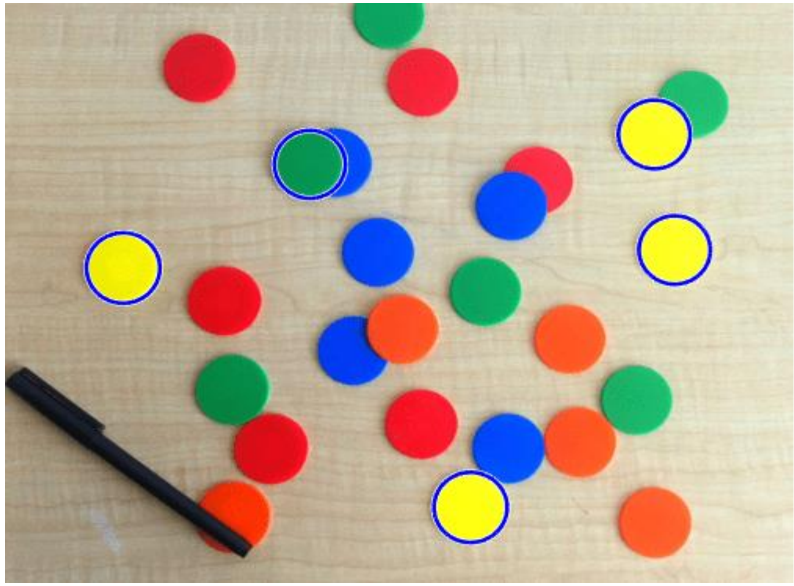
OpenCV

- 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

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 colors
2. We can use size to reject shapes
3. We can look for circular shapes in the image

Manchester Robotics

Color Filtering

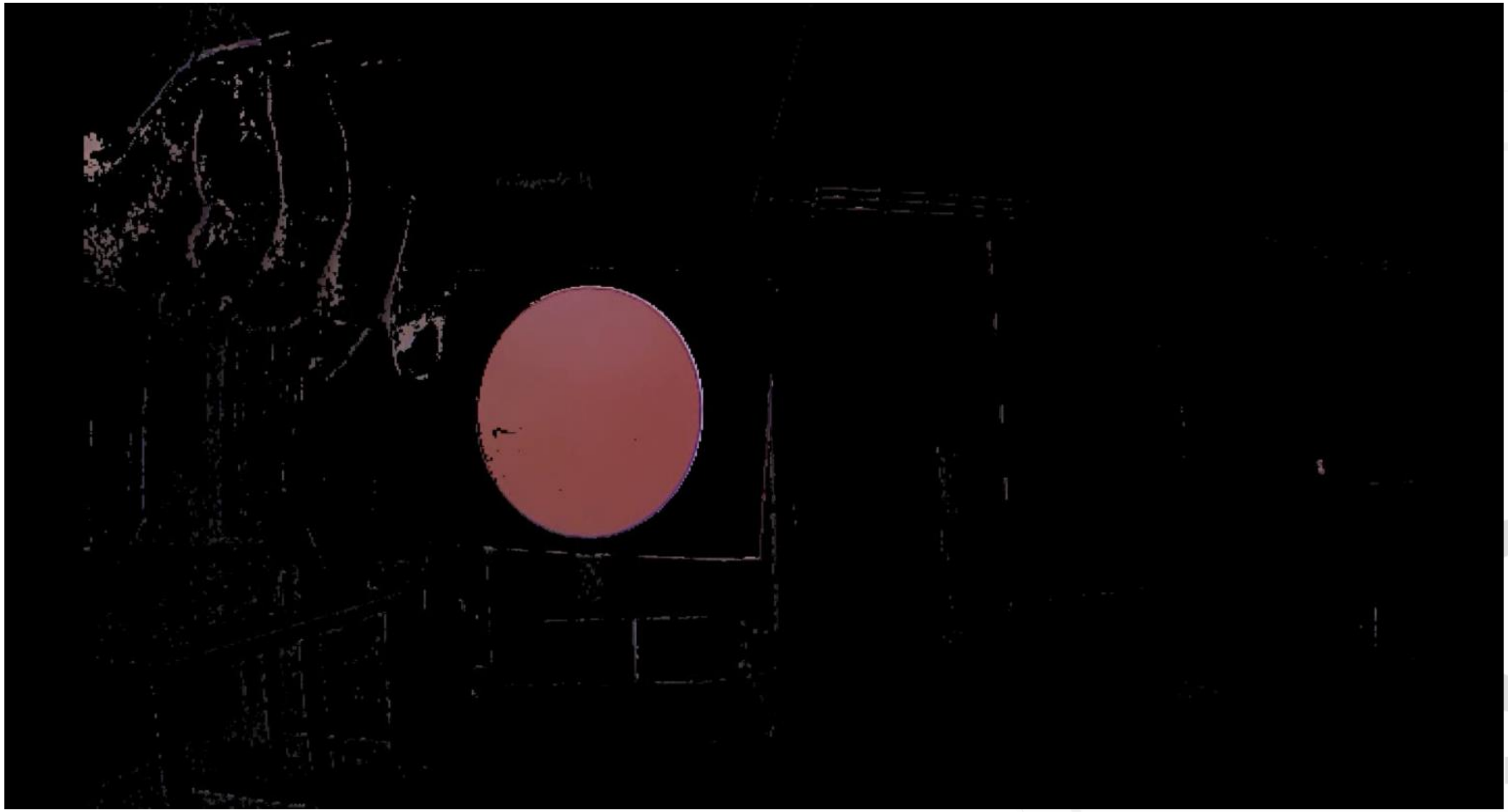
- Convert the base image to the hsv space
- Filter the image to remove all colors not defined as red or green, storing the results in two separate images
  - Use OpenCV's `inRange` function
  - Suggested values for the colors required

	RED	GREEN
H	0-33	49-98
S	88-255	39-255
V	179-255	130-255

https://docs.opencv.org/4.x/de/d25/imgproc_color_conversions.html

https://docs.opencv.org/3.4/da/d97/tutorial_threshold_inRange.html

Colour Filtering

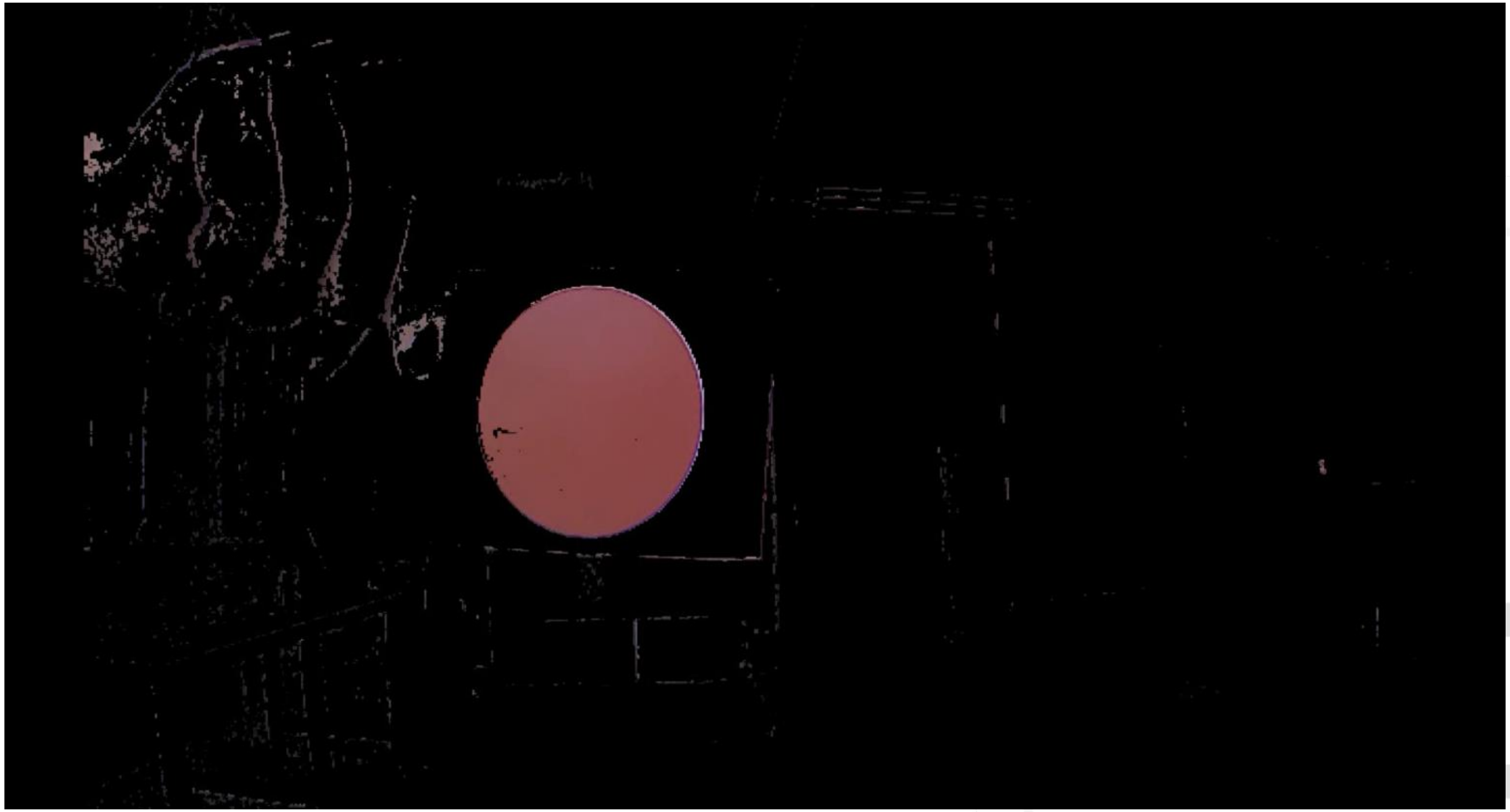


Thresholding

- Now to remove the background
- This is done with a simple binary threshold
  - use OpenCV's `threshold` function

https://docs.opencv.org/4.x/d7/d4d/tutorial_py_thresholding.html

Thresholding



OpenCV

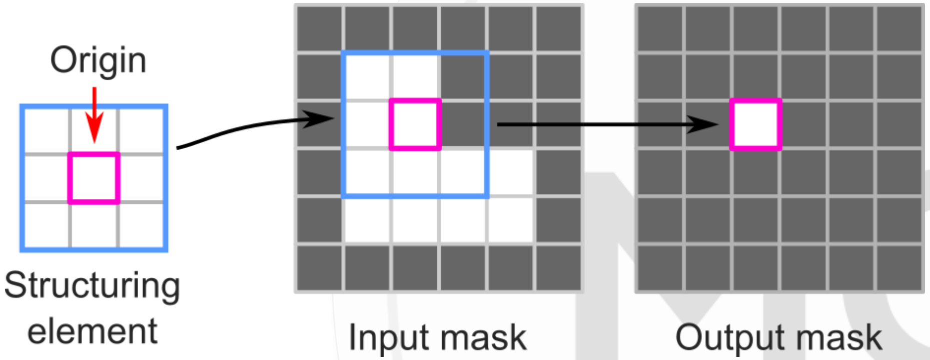
Noise rejection in color filtering

It is common that artifacts appear in the images after applying certain operations.

Noisy image

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

One of the most common techniques to deal with this artifacts is known as a morphological operators. Morphological operators are defined as a combination between an image and a structuring element.

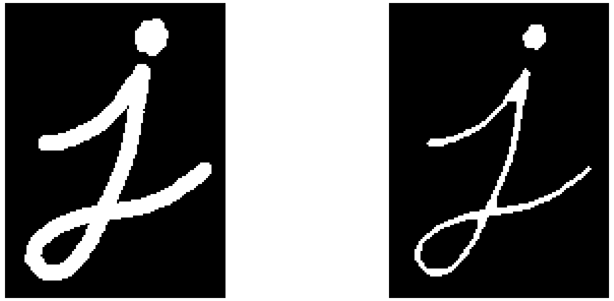


OpenCV

Noise removal tools

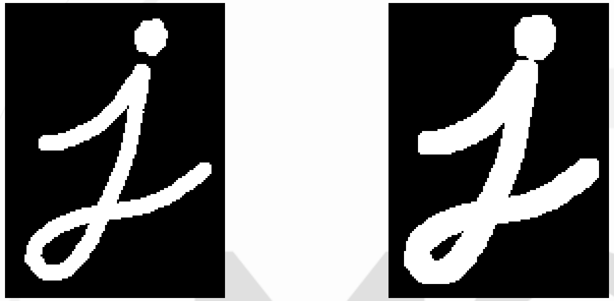
Erode

Using a mask (kernel) we compute the minimal value of a given area around a pixel and we replace it by that value. The results of this technique are shrinking the larger regions and removing the smallest ones.



Dilate

We convolute each pixel of an image with a given kernel, resulting in shapes becoming larger and smoothing their edges



OpenCV

Using morphological operators in OpenCV

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

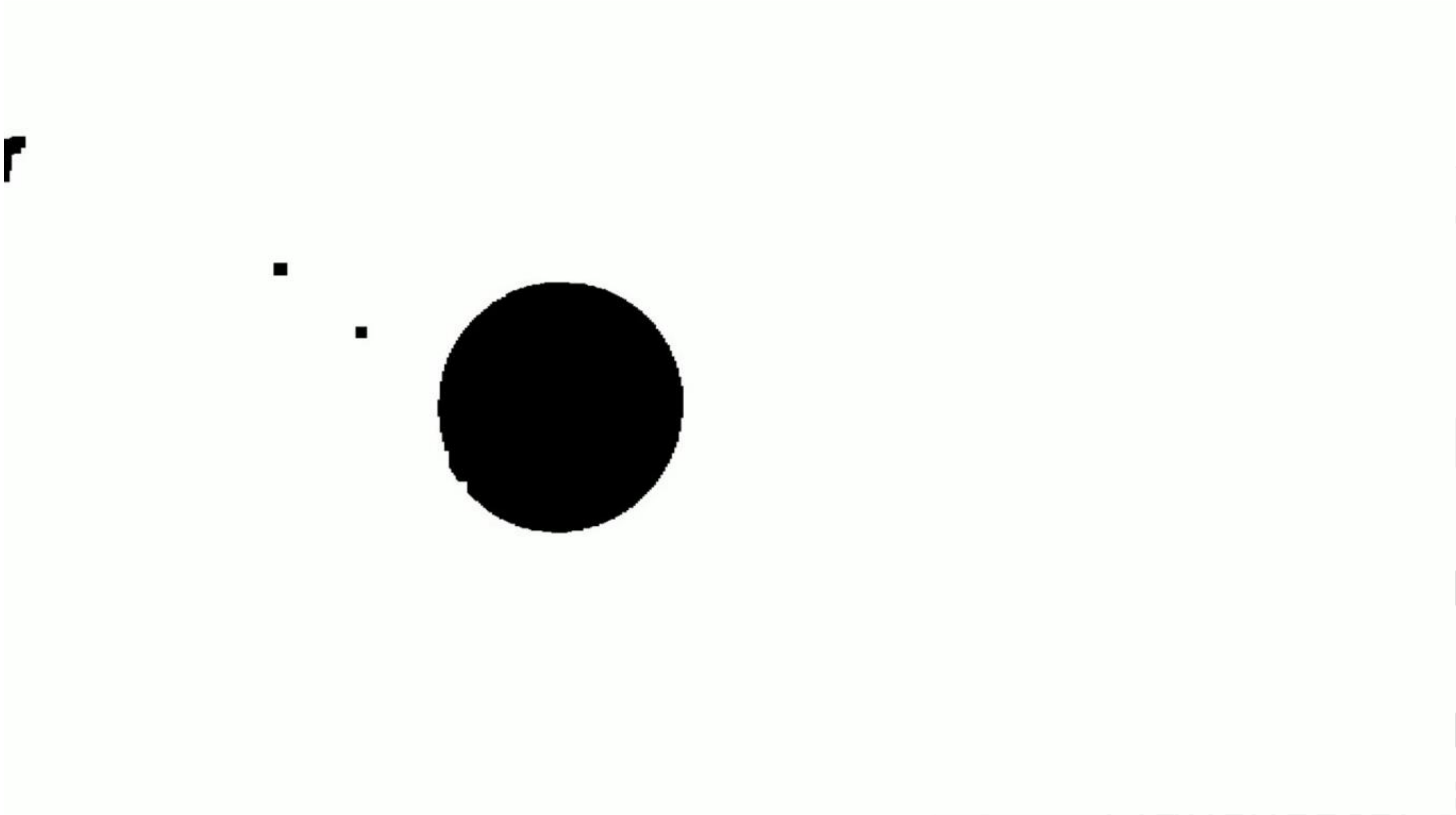
1. *The mask:* it will define the area of effect of our operation, for example in an erosion the bigger the mask the smaller will the final shape be.
2. *The operation:* OpenCV has functions defining the most common operators, so it is a matter of selecting the suitable option
3. *The number of iterations:*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

You can find more details about how this technique is used in the documentation link below.

Documentation:

https://docs.opencv.org/3.4/db/df6/tutorial_erosion_dilatation.html

Noise Removal



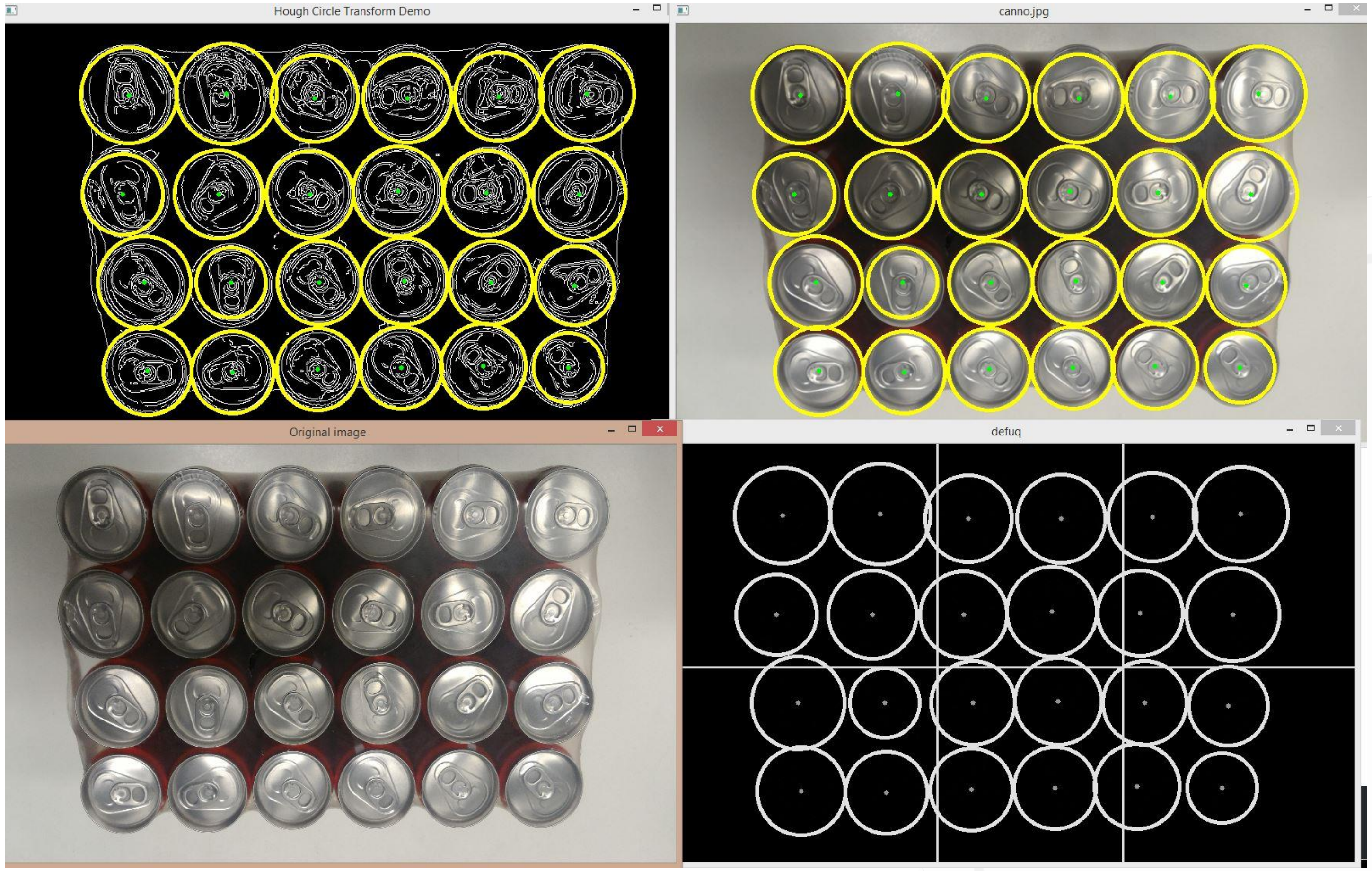
Hough Circles

- Technique used to detect circles. Potential candidates are “voted”.

$$(x-a)^2+(y-b)^{\,2}=r^{\,2}$$

- Several circles are proposed around a center point (a, b).
- The intersection point of *N* circles is “voted” as the original circle.

Hough Circles



Activity – Detect circles

- Detect circles
  - DetectCirclesExample_01.png

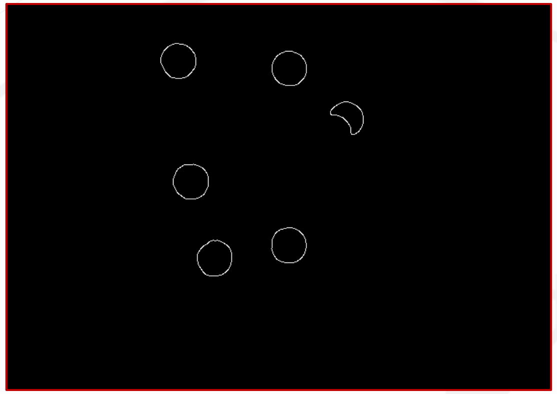
- Check if the image is in range
- Create a mask
- Remove noise
- Change to greyscale
- Detect circles

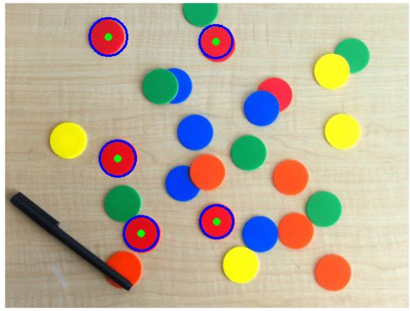
Activity – Detect circles

```
import cv2
import numpy as np

image = cv2.imread('DetectCirclesExample_01.png')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
#blur = cv2.medianBlur(gray, 5)
#blur = gray
blur =cv2.Canny(gray, 75, 250)
cv2.imshow('Blur', blur)
cv2.waitKey(0)
cv2.destroyAllWindows()

circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, 1.2, 50, param1=50, param2=30, minRadius=0, maxRadius=50)
#circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, 3, 50, param1=125, param2=150, minRadius=15, maxRadius=50)
circles = np.squeeze(circles).astype(int)
for i in circles:
    # draw the outer circle
    cv2.circle(image,(i[0], i[1]), i[2], (255, 0, 0), 2)
    cv2.circle(image, (i[0], i[1]), 2, (0, 255, 0), 5)

cv2.imshow('detected circles', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

Activity – Detect circles

- Detect red, green, and yellow circles
  - DetectCirclesExample_01.png
- Select a color and get its HSV value
  - https://www.developmenttools.com/color-picker/
- Check if the image is in range
- Create a mask
- Remove noise
- Change to greyscale
- Detect circles



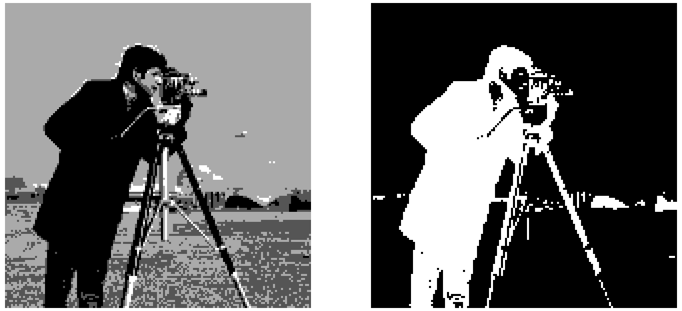


Activity – Detect colored circles

```
import cv2
import numpy as np

image = cv2.imread('DetectCirclesExample_01.png')

# green
lower = np.array([61, 67, 73], np.uint8)
upper = np.array([102, 255, 255], np.uint8)

hsvFrame = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

mask = cv2.inRange(hsvFrame, lower, upper)
detected_output = cv2.bitwise_and(image, image, mask =  mask)

gray = cv2.cvtColor(detected_output, cv2.COLOR_BGR2GRAY)
gray = cv2.medianBlur(gray, 5)
#blur = gray
blur = cv2.Canny(gray, 75, 250)

cv2.imshow('Blur', blur)
cv2.waitKey(0)
cv2.destroyAllWindows()

circles = cv2.HoughCircles(blur, cv2.HOUGH_GRADIENT, 1.2, 50, param1=10, param2=30, minRadius=0, maxRadius=50)
circles = np.squeeze(circles).astype(int)
for i in circles:
    cv2.circle(image,(i[0], i[1]), i[2], (255, 0, 0), 2)
    cv2.circle(image, (i[0], i[1]), 2, (0, 255, 0), 5)

cv2.imshow('detected circles', image)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

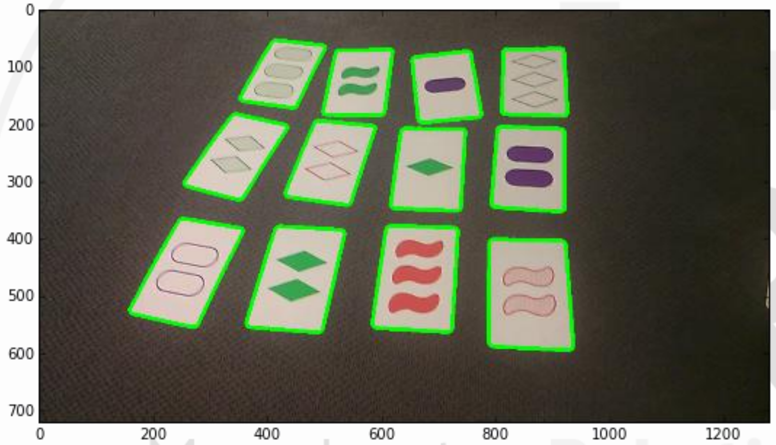
Threshold

- Function applied to a greyscale image
- A threshold is defined
  - 0 for smaller values
  - 1 for bigger values
- The binary image can have several formats



Contours

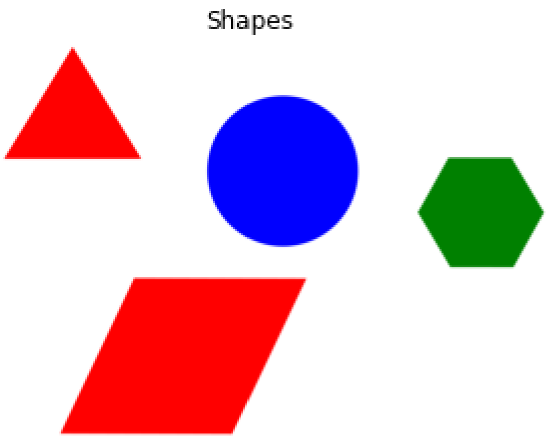
- Continuous set of points along a given boundary with a similar color or intensity
- High contrast images are required
- Contour retrieval types:
  - List, extreme outer flags, external and internal, hierarchical
- Contour approximation
  - Full list of points or vertices

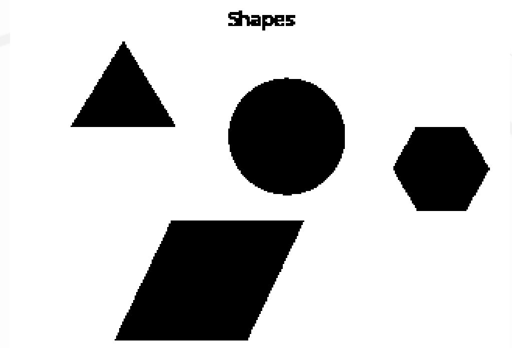


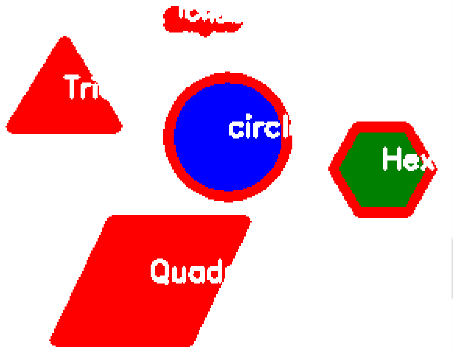
Manchester Robotics

Activity – Shape identification

- Change to greyscale
- Obtain threshold
- Find contours
- For each contour
  - Approximate the shape to *n* vertices
  - Find the center of the point shape







MCR2
Manchester Robotics

Activity

```
import cv2
import numpy as np
from matplotlib import pyplot as plt

img = cv2.imread('shapes.png')
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
_, threshold = cv2.threshold(gray, 127, 255, cv2.THRESH_BINARY)

contours, _ = cv2.findContours(threshold, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cv2.imshow('shapes', threshold)
cv2.waitKey(0)
cv2.destroyAllWindows()

i = 0
for contour in contours:
    if i == 0:
        i = 1
        continue

    approx = cv2.approxPolyDP(contour, 0.01 * cv2.arcLength(contour, True), True)
        cv2.drawContours(img, [contour], 0, (0, 0, 255), 5)

    M = cv2.moments(contour)
    if M['m00'] != 0.0:
        x = int(M['m10']/M['m00'])
        y = int(M['m01']/M['m00'])

    if len(approx) == 3:
        cv2.putText(img, 'Triangle', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
    elif len(approx) == 4:
        cv2.putText(img, 'Quadrilateral', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
    elif len(approx) == 5:
        cv2.putText(img, 'Pentagon', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
    elif len(approx) == 6:
        cv2.putText(img, 'Hexagon', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)
    else:
        cv2.putText(img, 'circle', (x, y), cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255, 255, 255), 2)

cv2.imshow('shapes', img)
cv2.waitKey(0)
cv2.destroyAllWindows()
```

{Learn, Create, Innovate};

OpenCV + ROS

Node integration

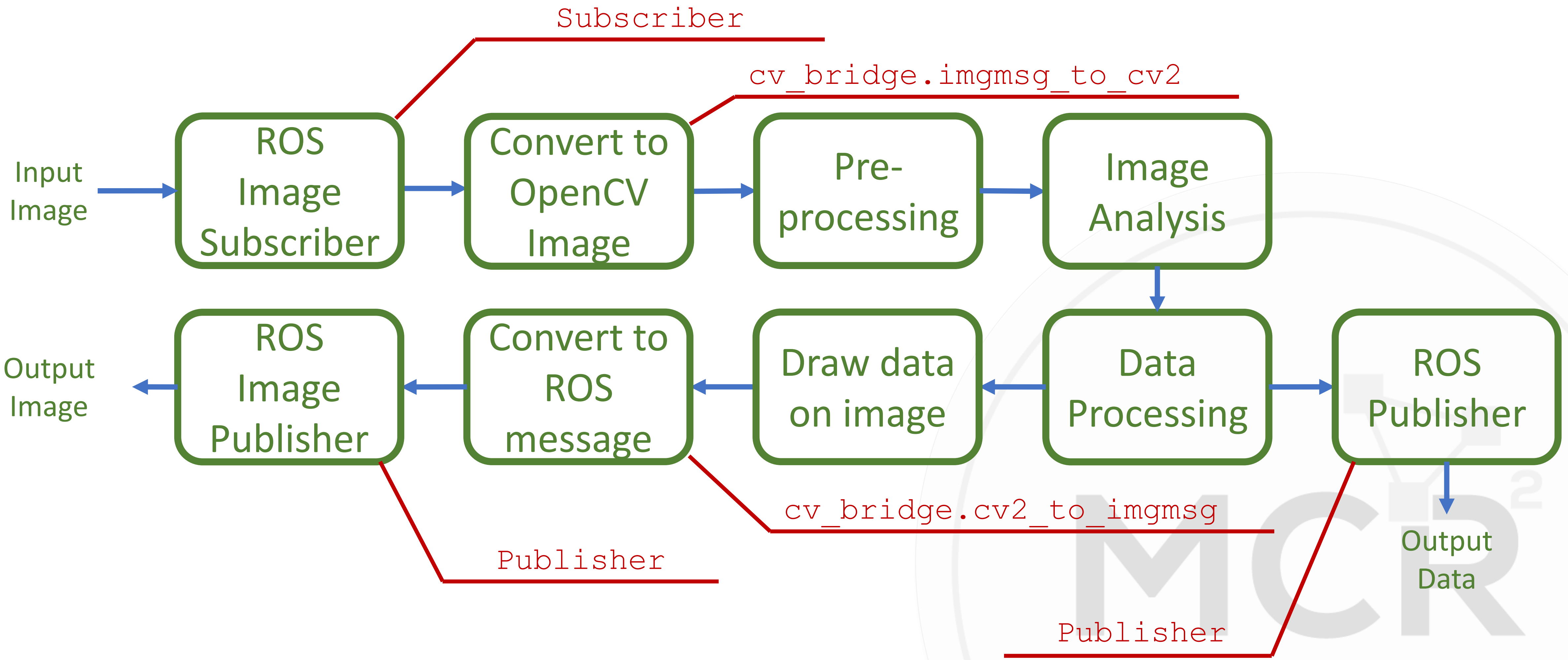
WSL – Camera integration

- usb - Capturing webcam video with OpenCV in WSL2 - Ask Ubuntu
- PINTO0309/wsl2_linux_kernel_usbcam_enable_conf: Configuration file to build the kernel to access the USB camera connected to the host PC using USBIP from inside the WSL2 Ubuntu 20.04/22.04. (github.com)
- How do I get the current username in Windows PowerShell? - Stack Overflow

Interfacing with ROS

- OpenCV comes preinstalled on the Jetson
- ROS uses a package called CV Bridge to interface with OpenCV:
  - `sudo apt install ros-humble-cv-bridge`
- To enable OpenCV in the ROS environment on the Jetson, add the following to the CMakeLists.txt in the package where OpenCV is required:
  - `set(OpenCV_DIR /usr/share/OpenCV)`
  - `find_package(catkin REQUIRED COMPONENTS(OpenCV)`
- Once this line is added, OpenCV methods can be imported into python using:
  - `import cv2`

Image Processing Node



Terminal

- Create a new package

```
ros2 pkg create –build-type ament_python –license Apache-2.0
  open_cv_example –dependencies rclpy sensor_msgs –node-name circle_id
```

- Install the required library to run a node for your webcam

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

- Edit the circle_id.py program

Image Processing Node

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

class ColorId(Node):
    def __init__(self):
        super().__init__('circle_id_node')

        self.img = None
        self.hsvFrame = None
        self.bridge = CvBridge()

        self.lower = np.array([136, 87, 111], np.uint8)
        self.upper = np.array([180, 255, 255], np.uint8)

        dt = 0.1

        self.subscription = self.create_subscription(Image, '/image_raw', self.camera_callback, 10)

        self.timer = self.create_timer(dt, self.timer_callback)
        self.circle_id_pub = self.create_publisher(Image, '/image_processing/color_id', 10)

        self.get_logger().info('Color identification node started!')
```

Image Processing Node

```
 def camera_callback(self, msg):
    try:
        self.img = self.bridge.imgmsg_to_cv2(msg, "bgr8")
    except:
        self.get_logger().info('Failed to convert image to CV2')

def timer_callback(self):
    hsvFrame = cv2.cvtColor(self.img, cv2.COLOR_BGR2HSV)
    mask = cv2.inRange(hsvFrame, self.lower, self.upper)
    detected_output = cv2.bitwise_and(self.img, self.img, mask =  mask)
    gray = cv2.cvtColor(detected_output, cv2.COLOR_BGR2GRAY)
    blur = cv2.medianBlur(gray, 5)
    canny = cv2.Canny(blur, 75, 250)

    try:
        circles = cv2.HoughCircles(canny, cv2.HOUGH_GRADIENT, 1.2, 50, param1=10, param2=30, minRadius=0, maxRadius=50)
        circles = np.squeeze(circles).astype(int)

        img = self.img

        for i in circles:
            cv2.circle(img,(i[0], i[1]), i[2], (255, 0, 0), 2)
    except:
        img = self.img

    self.circle_id_pub.publish(self.bridge.cv2_to_imgmsg(img, encoding='bgr8'))
```

Image Processing Node

```
def main(args=None):
    rclpy.init(args=args)
    c_id = ColorId()
    rclpy.spin(c_id)
    c_id.destroy_node()
    rclpy.shutdown()

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

Terminal

- Start the camera

```
ros2 run usb_cam usb_cam_node_exe
```

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

```
cd ~/ros2_ws/
colcon build
source install/setup.bash
ros2 run open_cv_example circle_id
```

- Open the image viewer in a new terminal and validate your results

```
ros2 run rqt_image_view rqt_image_view
```

Thank You

Robotics For Everyone

{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.*