

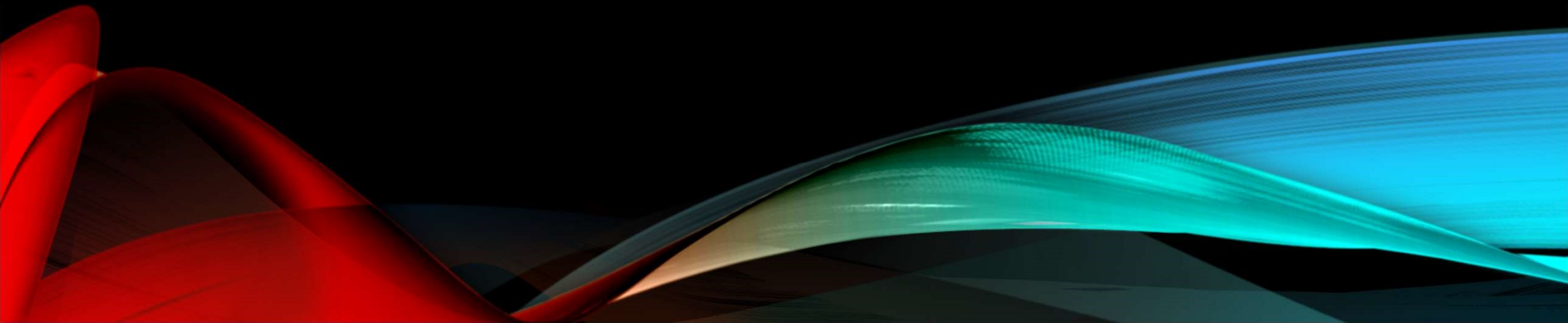
QUT ROBOTICS CLUB OPENCV WORKSHOP

Introduction to robotic vision with
OpenCV in python

Covering Computer Vision
concepts and integrating
computer vision with robotic
action

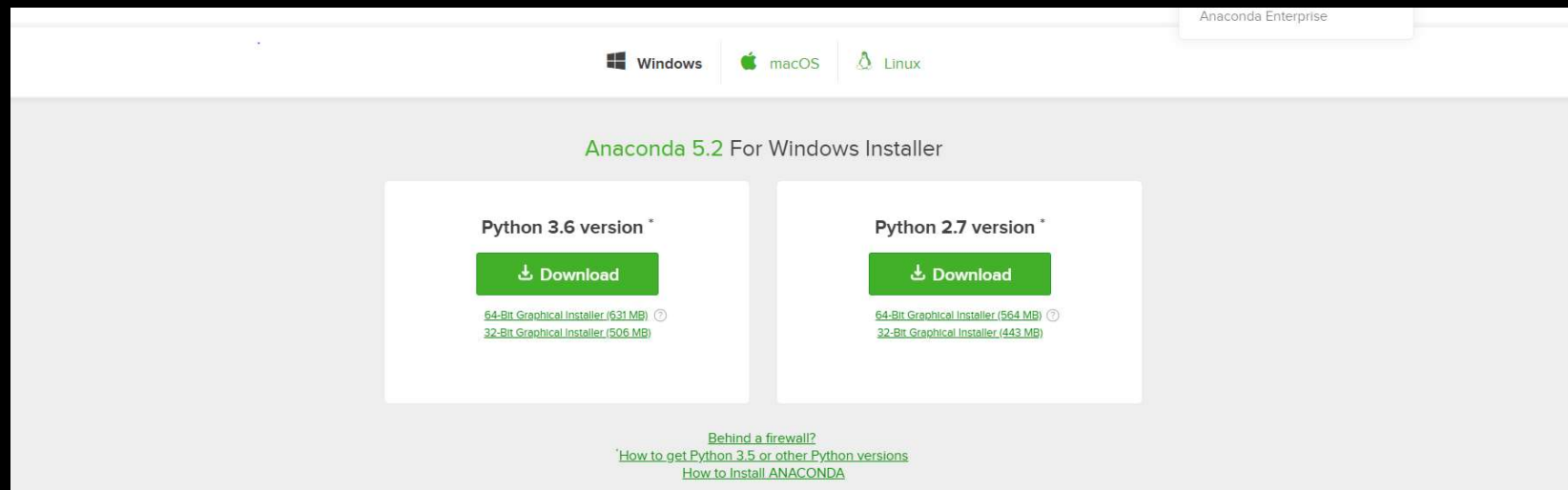


GETTING OPENCV IN PYTHON ON A COMPUTER WITH ANACONDA



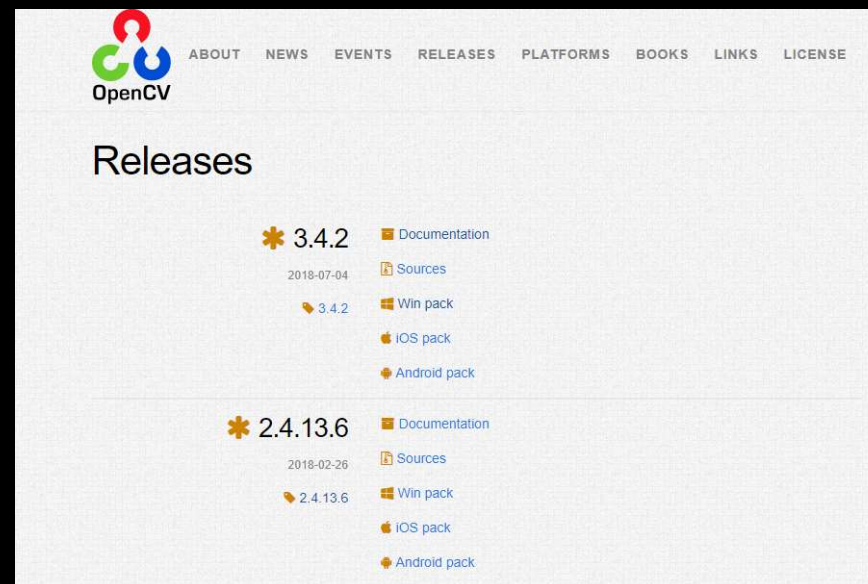
INSTALLING OPENCV ONLY FOR PYTHON PRACTICE ON A COMPUTER

- Search for Anaconda online to download the Spyder IDE for Python development: Pick **Python 2.7** for OpenCV!
- <https://www.anaconda.com/download/#windows>
- Some University computers may already have this one installed



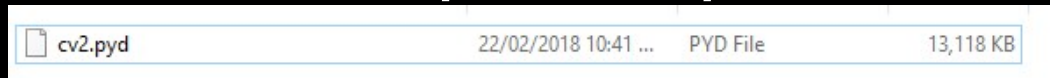
INSTALLING OPENCV ONLY FOR PYTHON PRACTICE ON A COMPUTER

- Once you have Python 2.7 working on Spyder, You can go get OpenCV!
- <https://opencv.org/releases.html>



INSTALLING OPENCV ONLY FOR PYTHON PRACTICE ON A COMPUTER

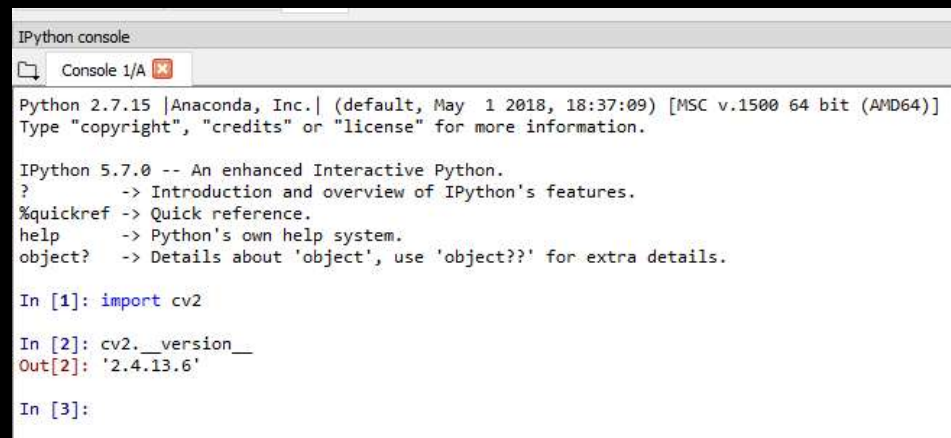
- Go find the OpenCV package (cv2.pyd) from the OpenCV you downloaded/ extracted:
- ***C:\Users\...\Documents\OpenCV\opencv\build\python\2.7\x64***



- Copy the file and paste it into the library site-packages for anaconda2 usually its in here:
- ***C:\Users\...\AppData\Local\Continuum\anaconda2\Lib\site-packages***

INSTALLING OPENCV ONLY FOR PYTHON PRACTICE ON A COMPUTER

- Once you've pasted it into site-packages, start up Spyder and in the console you should be able to import cv2



```
IPython console
Console 1/A
Python 2.7.15 |Anaconda, Inc.| (default, May 1 2018, 18:37:09) [MSC v.1500 64 bit (AMD64)]
Type "copyright", "credits" or "license" for more information.

IPython 5.7.0 -- An enhanced Interactive Python.
? -> Introduction and overview of IPython's features.
%quickref -> Quick reference.
help -> Python's own help system.
object? -> Details about 'object', use 'object??' for extra details.

In [1]: import cv2

In [2]: cv2.__version__
Out[2]: '2.4.13.6'

In [3]:
```

- Notice its important to have the right Python interpreter (Python2.7) using the module or otherwise it may not find it and give an error. Also note that the version stated is the release number on OpenCV's website.

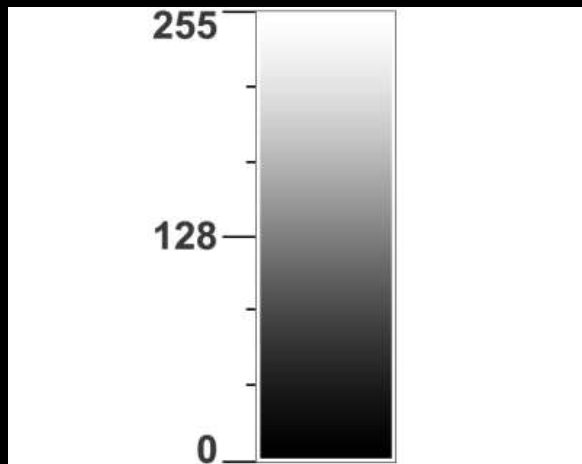
IMAGE PROCESSING FOR ROBOTIC VISION

Colour Thesholding



GREYSCALE IMAGES

- What is an image or a frame? It's a matrix with colour intensity values:



Intensity is usually an Uint8 unsigned integer 8 bit: (0 to 255) or a float (0 to 1)

	0	1	2	3	4
0	255	184	178	84	129
1	84	255	255	130	84
2	78	255	0	0	78
3	84	130	255	130	84

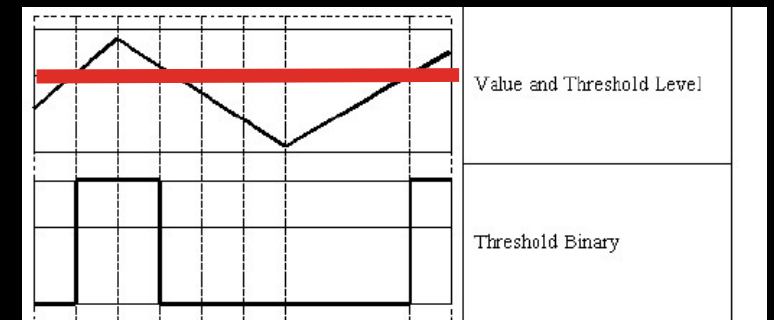
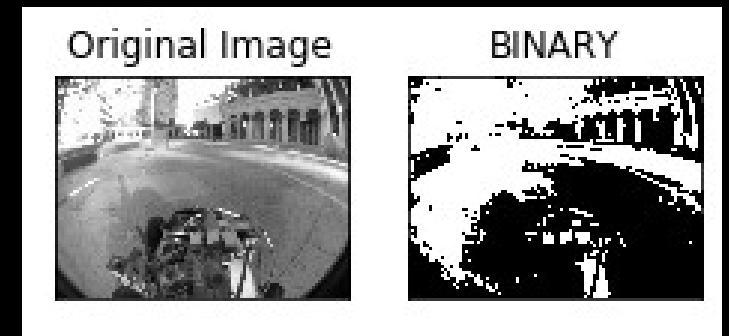
This is how a grayscale image is structured, it starts at u,v coordinate (0,0) at the top left corner. Each coordinate is a pixel



DRC back in the old days...

THRESHOLDING GREYSCALE IMAGES TO GET BINARY IMAGES

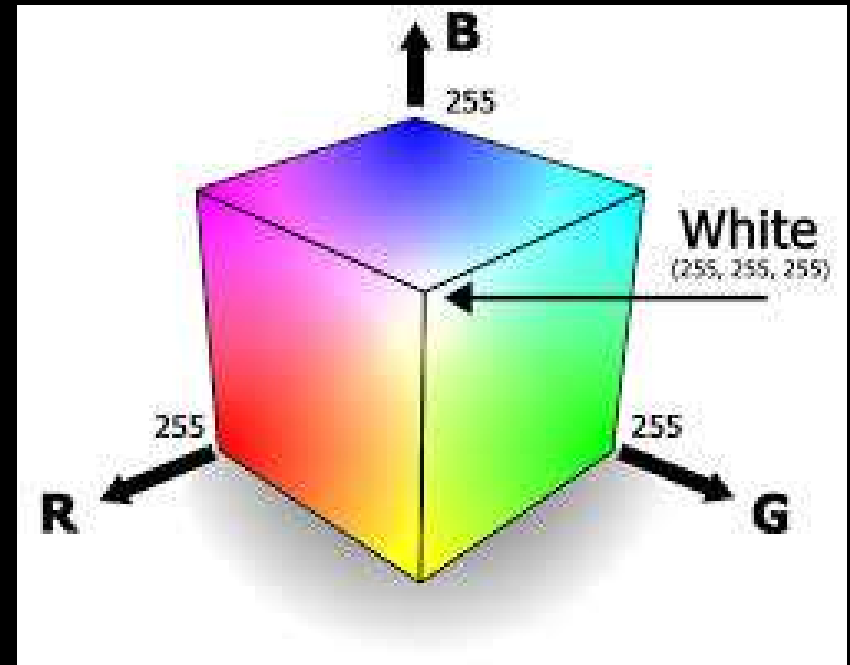
- Thresholding is a **Logic operation**, The idea is that we loop each pixel and test if the intensity is greater or less than the threshold value.
- If pixel value is greater than threshold -> answer is 1 (white) else it is 0 (black). It can be a range too. This gives us a binary image where each pixel is logical (0 or 1) that's (white or black)
- Usually We threshold with a range. If pixel value is between a lower threshold and upper threshold, make it 1 else make it 0
- `cv.threshold(image,lower,upper,cv.THRESH_BINARY)`
- https://docs.opencv.org/3.4.0/d7/d4d/tutorial_py_thresholding.html

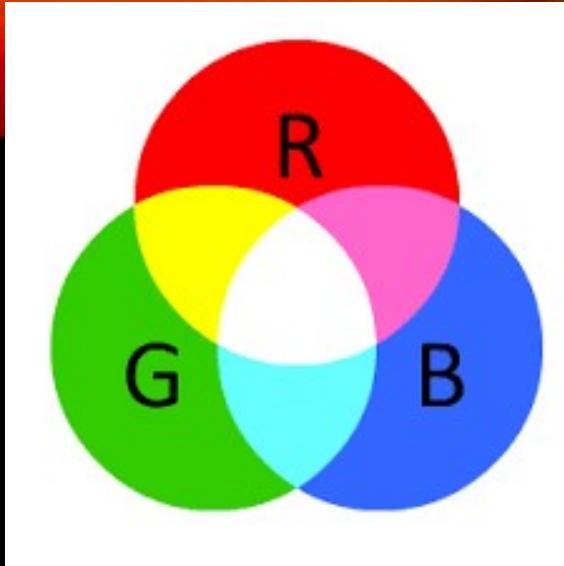


RED GREEN BLUE IMAGES

- A RGB image is a Red Green and Blue channel image.
- Its made of 3 layers of grayscale images. That is size = (width,height,3)
- You can separate them as planes like in the code below, notice that OpenCV structures it as Blue Green Red.
- In Python the first plane starts at 0 then 1 then 2

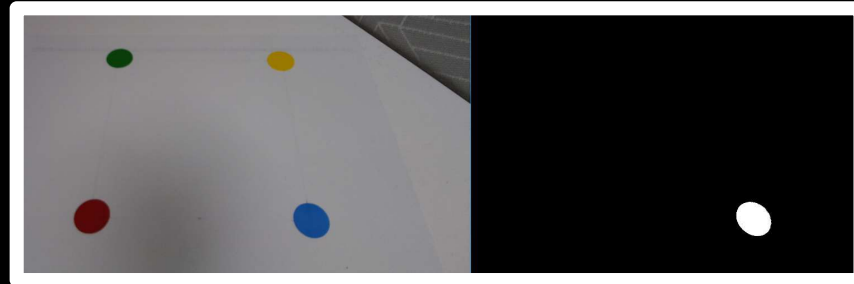
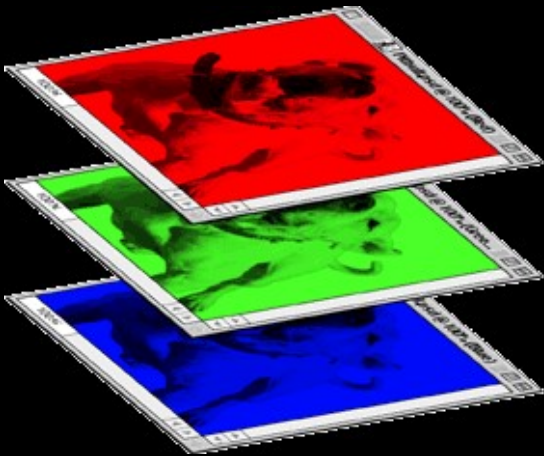
```
b, g, r = image[:, :, 0], image[:, :, 1], image[:, :, 2] # For RGB image
```





THRESHOLDING IN DIFFERENT COLOUR PLANES

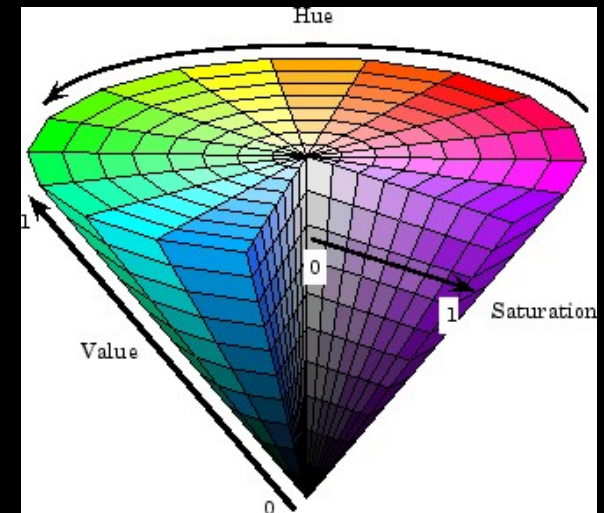
- Since in Colour each pixel is a BGR vector
- Pixel = [b , g , r]
- What we do is just do the same type of thresholding but for each colour plane
- Use `cv2.inRange(imageBGR7,lower,upper)`



- <https://medium.com/@ckyrkou/color-thresholding-in-opencv-91049607b06d>

HSV COLOUR SPACE

- RGB is a simple colour space suitable to storage and computing etc.
- But that colour maybe unintuitive to threshold or interpret for the Human eye. It may also be prone to brightness/darkness problems
- This is why there exists another Colour space called HSV (Hue Saturation and Value)
- HSV is designed to be more like a colour wheel where you only need to vary the Hue value to get the colour you want from the rainbow.



How to Use the HSV Color Model

The HSV color wheel is sometimes depicted as a cone or cylinder, but always with these three components:

Hue

Hue is the color portion of the color model, and is expressed as a number from 0 to 360 degrees:

Color	Angle
Red	0-60
Yellow	60-120
Green	120-180
Cyan	180-240
Blue	240-300
Magenta	300-360

Saturation

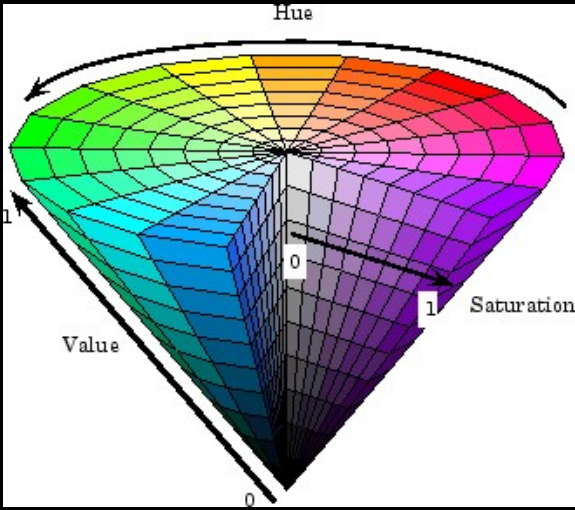
Saturation is the amount of gray in the color, from 0 to 100 percent. A faded effect can be had from reducing the saturation toward zero to introduce more gray.

However, saturation is sometimes viewed on a range from just 0-1, where 0 is gray and 1 is a primary color.

Value (or Brightness)

Value works in conjunction with saturation and describes the brightness or intensity of the color, from 0-100 percent, where 0 is completely black and 100 is the brightest and reveals the most color.

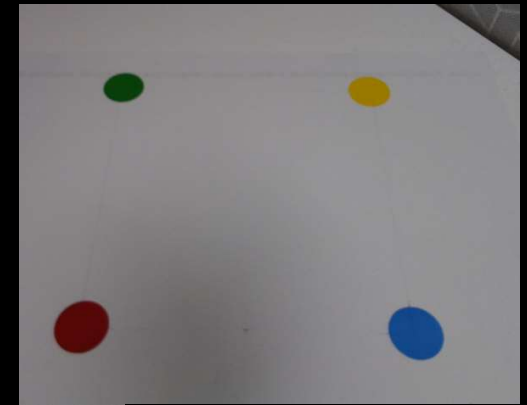
HSV COLOUR SPACE



NOTE: For HSV, Hue range is [0,179], Saturation range is [0,255] and Value range is [0,255]. Different software use different scales. So if you are comparing OpenCV values with them, you need to normalize these ranges. https://docs.opencv.org/3.4/df/d9d/tutorial_py_colorspaces.html

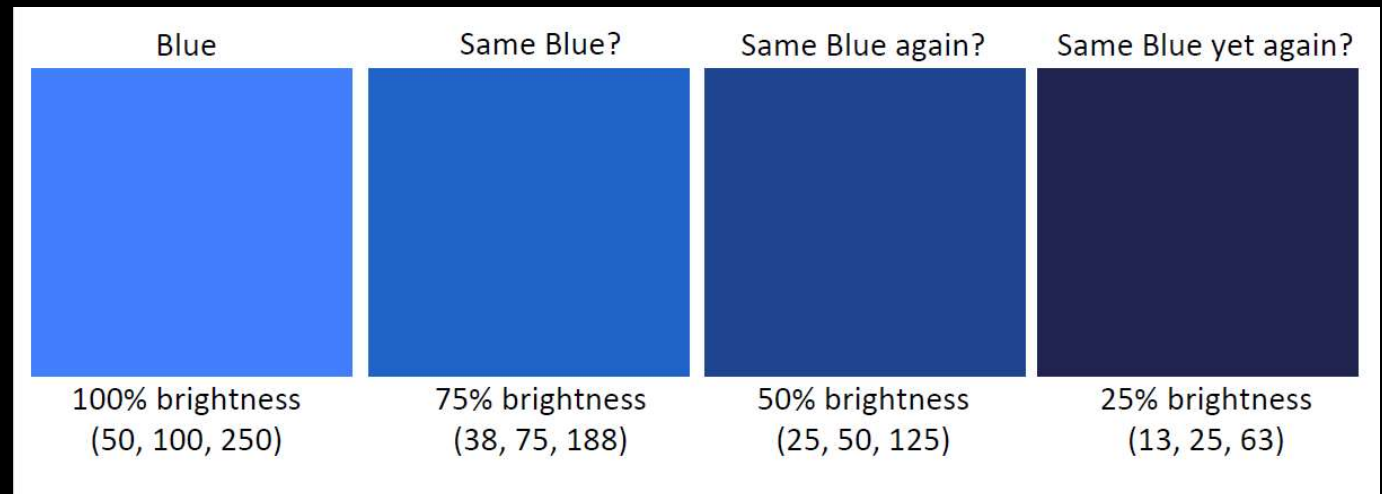
THRESHOLDING IN HSV EXAMPLE

- First convert BGR to HSV
- `hsv = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)`
- Then Do the `inRange` function with the HSV values
- #Threshold bounds note its (h,s,v) so $100 < h < 140$, $50 < s < 255$, $50 < v < 255$
- #Note Hue range is $[0, 179]$, Saturation range is $[0, 255]$ and Value range is $[0, 255]$
- `lower_bounds = np.array([100, 50, 50])`
- `upper_bounds = np.array([140, 255, 255])`
- `mask = cv2.inRange(hsv, lower_bounds, upper_bounds)`
- https://docs.opencv.org/3.4/df/d9d/tutorial_py_colorspaces.html



RGB - CHROMATICITY

- But there also a technique to fix the brightness problem
- The solution is to make each colour relative the overall brightness of the image
- sum the r,g,b matrices and divide each colour matrix by that sum:
- The image becomes a float from 0 to 1



$$r = \frac{R}{R + G + B}, g = \frac{G}{R + G + B}, b = \frac{B}{R + G + B}$$

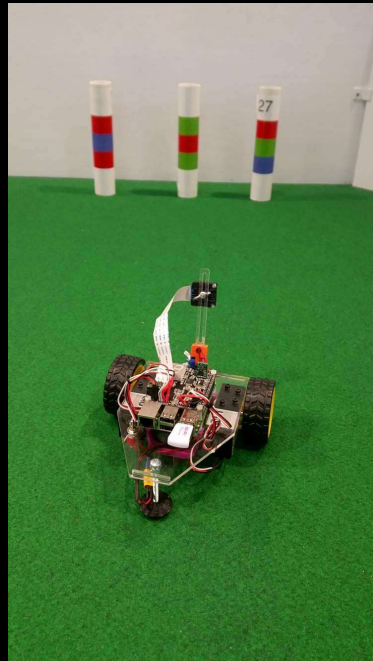
THRESHOLDING IN CHROMATICITY EXAMPLE

- To do chromaticity you may need to do things the old fashioned way, that's write a for loop to go around each pixel and compute the chromaticity
- <http://answers.opencv.org/question/54727/convert-rgb-brg-to-rg-chromaticity/>



TASK FOR THE WORKSHOP

- Your Task is to threshold a variety of images (or your own) from the arm project to images I retained from older subjects. I also found a DRC 2016 image.





NEXT WEEK'S OBJECTIVES

- Masks and morphological image processing with Binary images
- Edge Detection from Binary images
- Blob detection with binary images
- Extracting Features about blobs

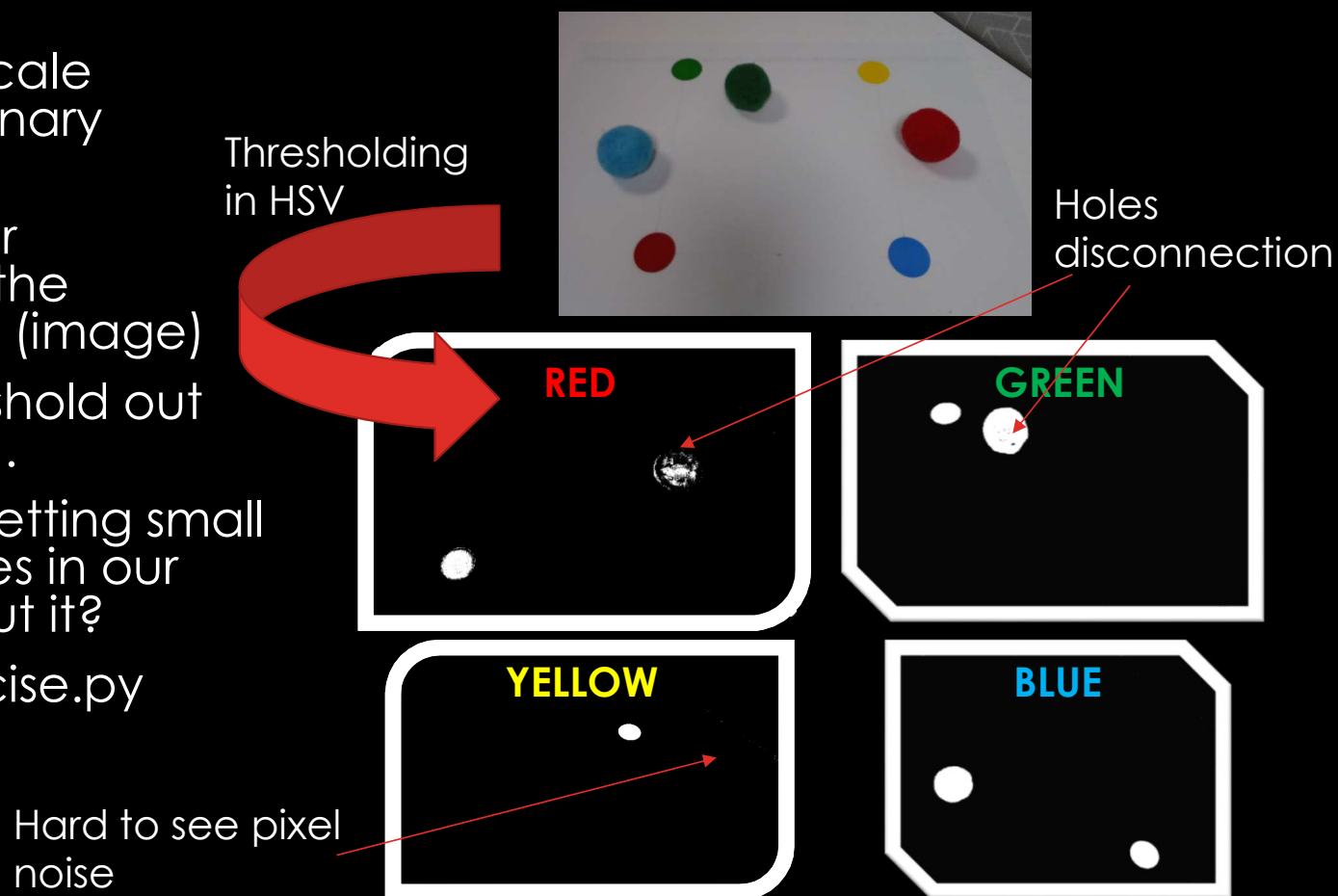
EXTRACTING FEATURES FOR ROBOTIC VISION

Blob detection from Binary Images



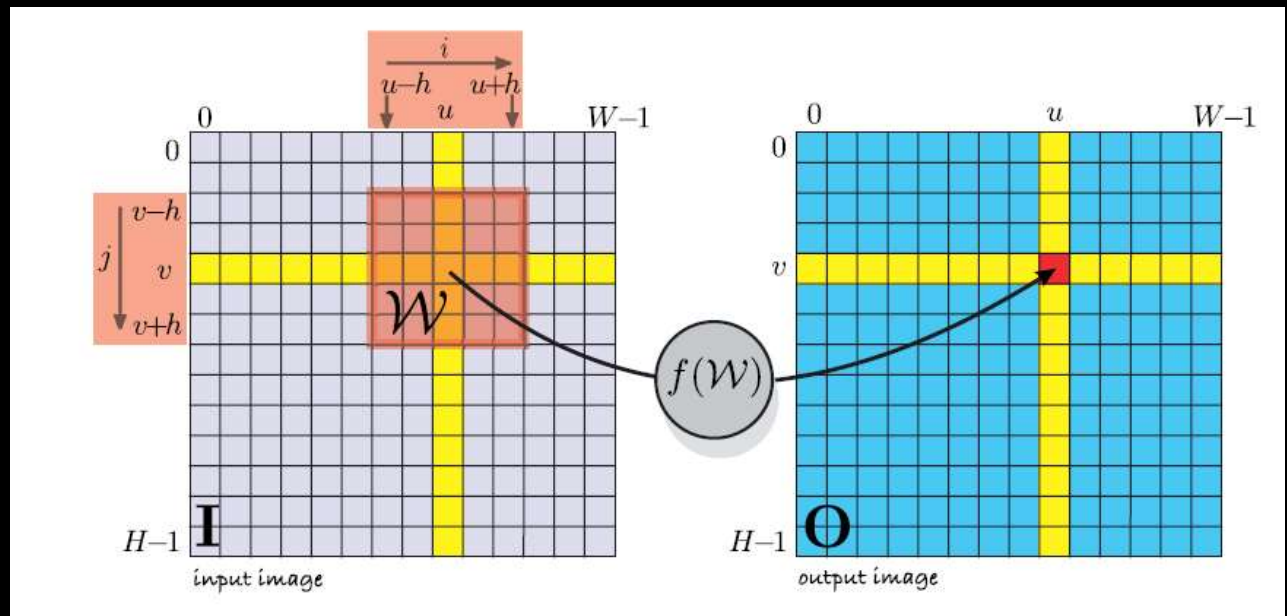
CONTINUING FROM LAST WEEK...

- We were introduced to grayscale images, colour images and binary images
- We can use either RGB, HSV or chromaticity to threshold out the coloured objects in the scene (image)
- Lets say we managed to threshold out Red, Blue, Green and Yellow...
- There is a problem we keep getting small scatters of colour or have holes in our objects what can we do about it?
- RUN `OpenCV-workshop-exercise.py`



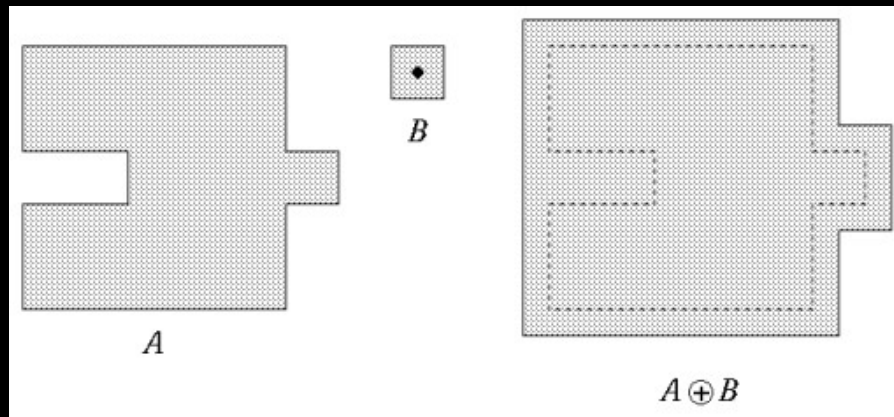
MORPHOLOGICAL IMAGE PROCESSING: A SPATIAL OPERATION

- Morphological image processing can be the answer.
- We use a spatial operation where each pixel becomes a function of the pixels in the window (or also called a kernel or a structuring element)
- Morphological Image processing is the case where we apply LOGIC Kernel functions in BINARY images.
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_morphological_ops/py_morphological_ops.html



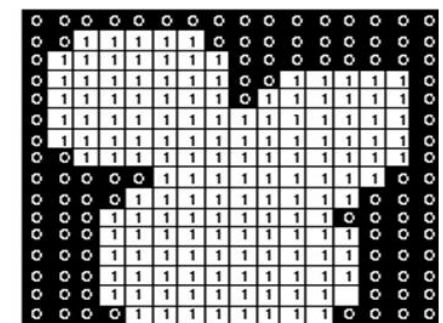
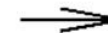
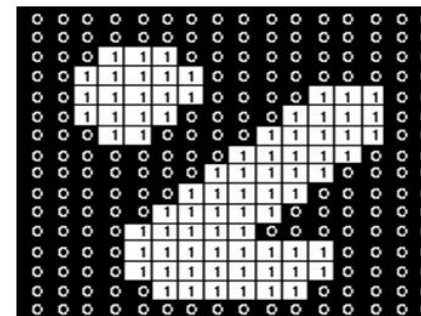
DILATION: MORPHOLOGICAL PROCESS

- For each pixel, the kernel does an OR operation. That means if ANY value in the pixel region (like the 3x3 kernel in the figure) is 1 then the result is 1 else its 0.
- This has the effect of making white thresholds bigger



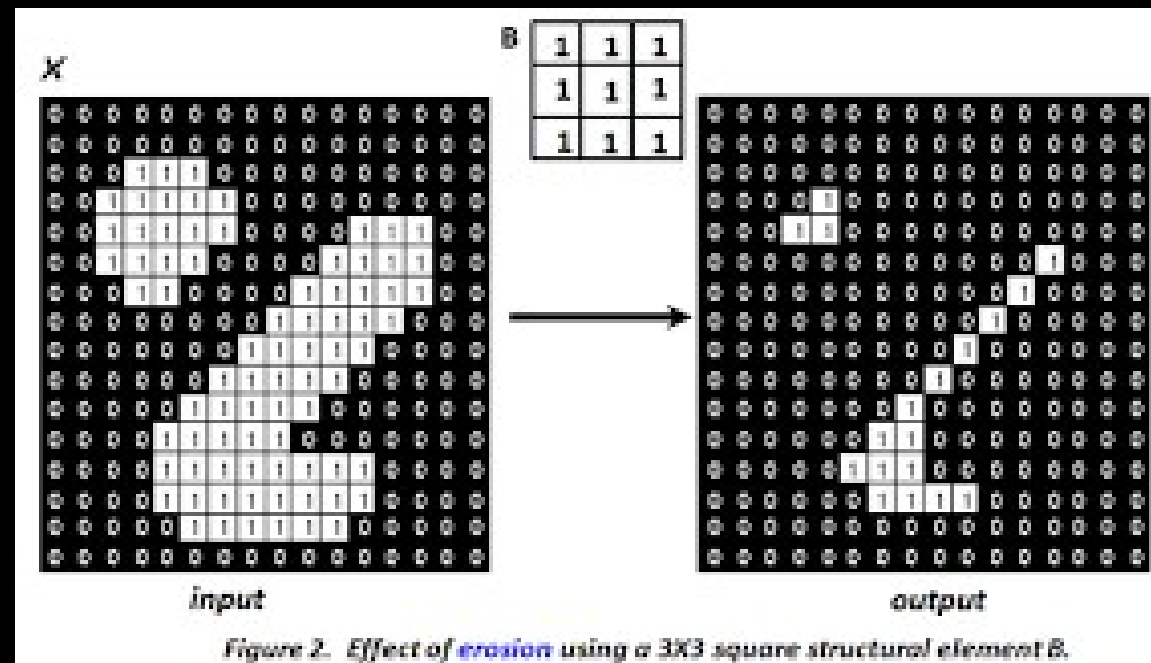
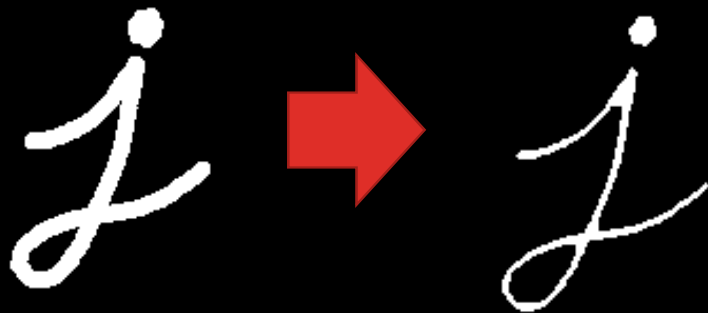
Morphological Image processing

- Dilation by example:
 - B: a 3×3 mask.



EROSION: MORPHOLOGICAL PROCESS

- For each pixel, the kernel does an AND operation. That means if ALL values in the pixel region (like the 3x3 kernel in the figure) is 1 then the result is 1 else its 0.
- This has the effect of shrinking white thresholds and removing any group of pixels smaller than the kernel.



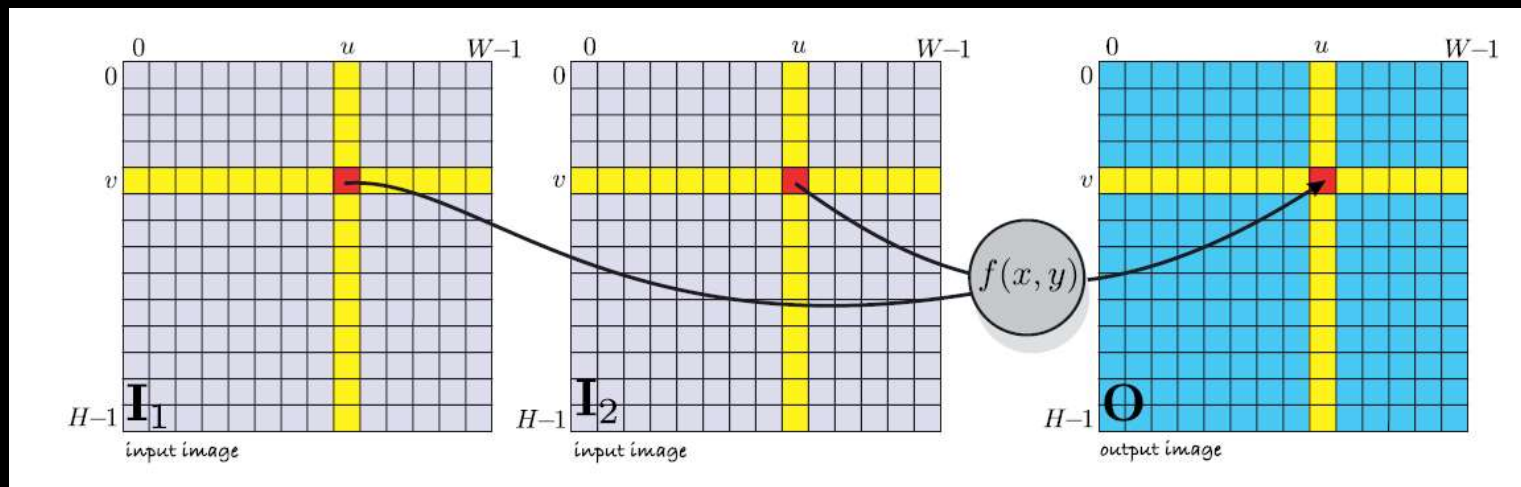
OPENING AND CLOSING

- We can use these processes to remove thresholding noise and fill in gaps our thresholding failed to do...
- Opening does Erosion first then Dilation. This kills small noise and retains the shape of the big thresholds.
- Closing does Dilation first then Erosion. This partially fills in gaps and holes in the thresholds while retaining the shape we want.



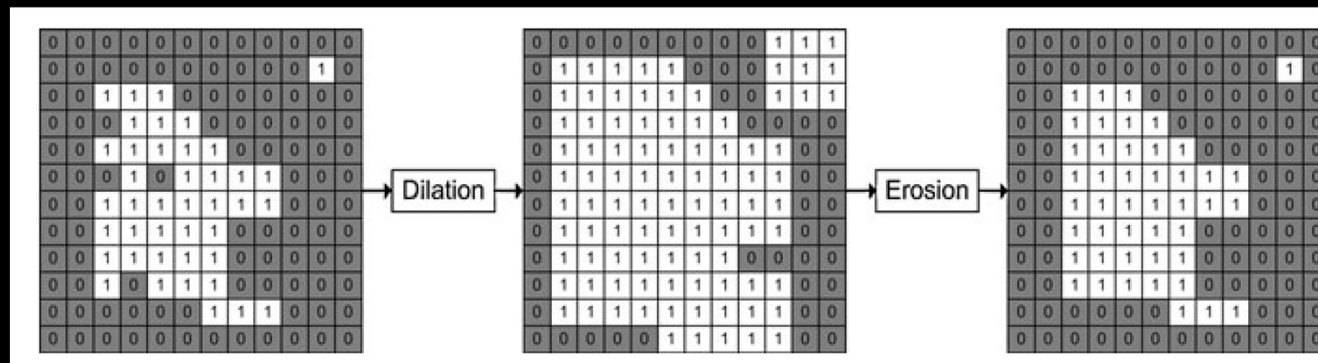
GETTING AN EDGE OR BOUNDARY WITH A DIADIC PROCESS

- We can also use dilation and erosion to get an outline of a shape.
- We do this by first dilation in I_1 and erosion in I_2 , we then do subtraction $I_1 - I_2$ to get an image with the outline:



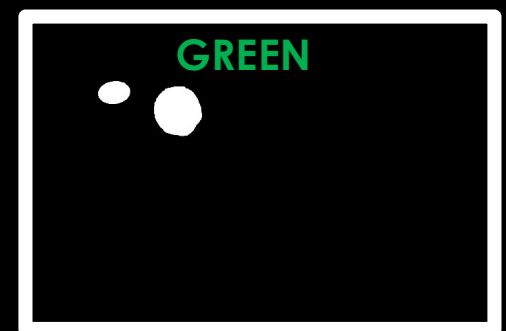
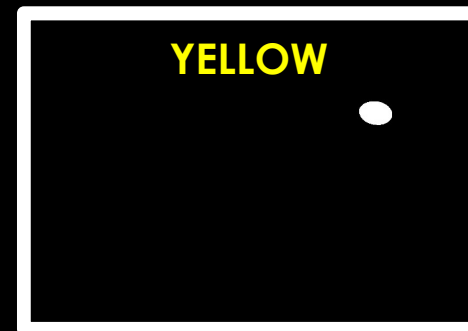
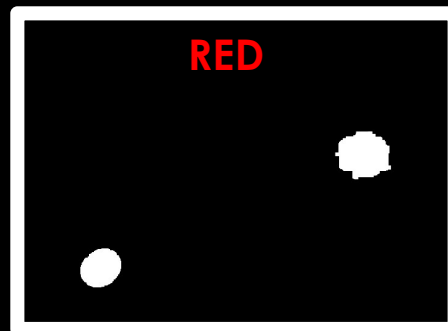
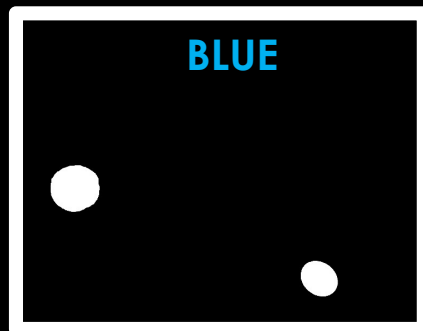
DOING THIS IN OPENCV:

- Kernel initialisation: `kernel = np.ones((5,5),np.uint8) #Gives a 5x5 kernel`
- Dilation: `dilation = cv2.dilate(img,kernel,iterations = 1)`
- Erosions: `erosion = cv2.erode(img,kernel,iterations = 1)`
- Opening: `opening = cv2.morphologyEx(img, cv2.MORPH_OPEN, kernel)`
- Closing: `closing = cv2.morphologyEx(img, cv2.MORPH_CLOSE, kernel)`
- Edge: `gradient = cv2.morphologyEx(img, cv2.MORPH_GRADIENT, kernel)`



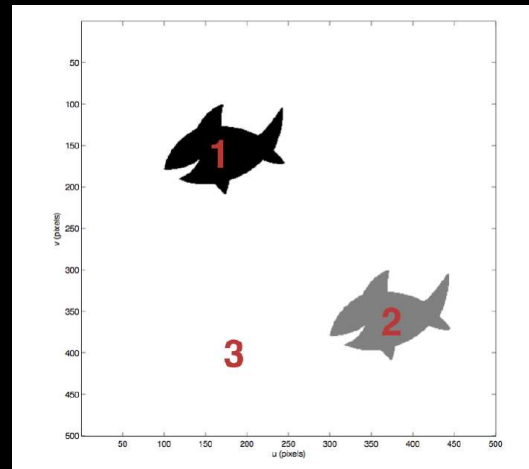
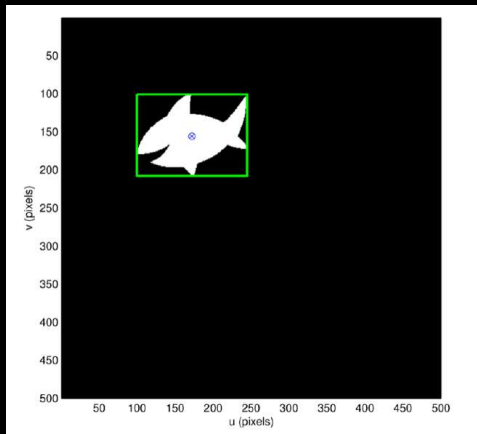
BINARY IMAGE BLOB DETECTION

- Now we have a binary image that clearly defines the objects we are segmenting – ‘figuring out where they are in the image’
- The last thing we will cover is how to recognise features about the thresholded binary images: We use Blob detection



BLOB ALGORITHM:

- We do want a connectivity analysis with the image. That is we check which same value pixels are next to each other. It can be 4 connectivity or 8 connectivity: where 8 connects diagonally the same pixels. The algorithm outputs a list of keypoints that correspond to the number white threshold 'blobs' in the image.



- Connectivity analysis is used to see which pixels are connected (i.e. are part of the same blob)

1	1	1
0	0	1
1	1	0

- For each pixel in the image, check the adjoining pixels
- Connectivity of four**
 - Are the pixels to the left, right, above and below connected?

- The second method is to check all adjoining pixels

1	1	1
0	0	1
1	1	0

- Connectivity of eight**
 - Are the pixels to the left, right, above, below and on all four diagonals connected?

FEATURE EXTRACTION FROM BLOBS (CENTROID, SHAPE AND AREA)

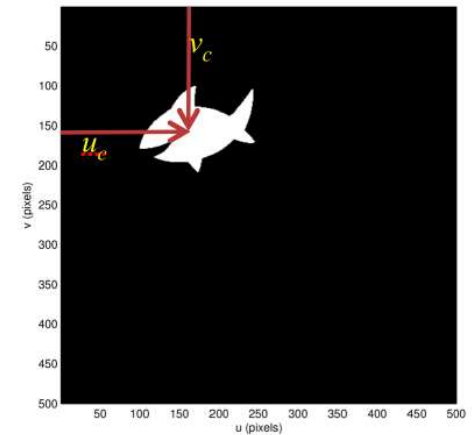
- From each blob, we can compute the centroid, area, circularity, convexity Inertia etc...
- The centroid is just the average coordinate point:
- Circularity tells you how circular the shape is from 0 to 1

$$\frac{4\pi Area}{(perimeter)^2}$$

- This is a powerful tool to recognise shapes,

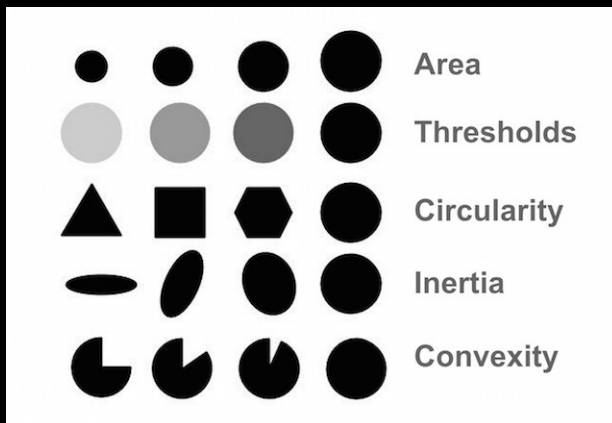
$$u_c = \text{Sum of } u \text{ values} / \text{Area}$$

$$v_c = \text{Sum of } v \text{ values} / \text{Area}$$



FEATURE EXTRACTION FROM BLOBS (CENTROID, SHAPE AND AREA)

- Link <https://www.learnopencv.com/blob-detection-using-opencv-python-c/>
- Here shows some info of blob features you can detect:
- OpenCV can even do Blob detection filtering out the blobs based on features.



```
#-----Setting up blob detection-----  
#  
# Setup SimpleBlobDetector parameters.  
params = cv2.SimpleBlobDetector_Params()  
  
# Change thresholds - accept all white blobs  
params.minThreshold = 0;  
params.maxThreshold = 255;  
  
# Filter by Area.  
params.filterByArea = False  
params.minArea = 1500  
  
# Filter by Circularity  
params.filterByCircularity = False  
params.minCircularity = 0.1  
  
# Filter by Convexity  
params.filterByConvexity = False  
params.minConvexity = 0.87  
  
# Filter by Inertia  
params.filterByInertia = False  
params.minInertiaRatio = 0.01  
  
# Set up the blob detector with parameters.  
detector = cv2.SimpleBlobDetector_create(params)
```

CODE ADVICE TO BLOB DETECTION:

- # Detect blobs.
- `red_mask = cv2.bitwise_not(red_mask)`
- **`keypoints = detector.detect(red_mask)`**
- `red_mask = cv2.bitwise_not(red_mask)` #Return the colours to normal

How we do blob detection. Note in OpenCV we need to invert the colours with `bitwise_not`

- #Check no blob detection case:
- `n = len(keypoints)`

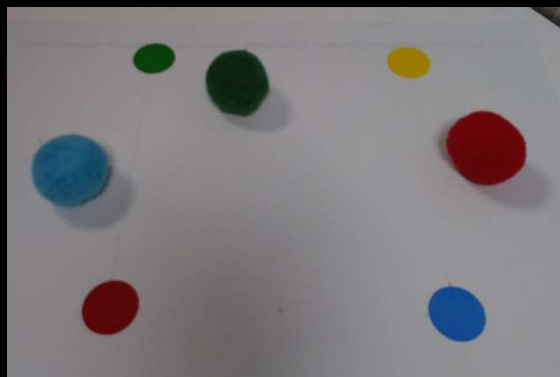
In a real system you need to take account of the possibility of there being no blob.

- if `n` is 0:
- `print('No red blobs detected')`
- else:

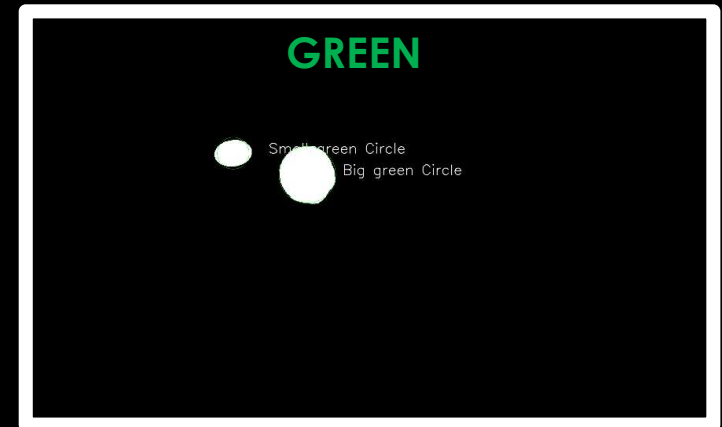
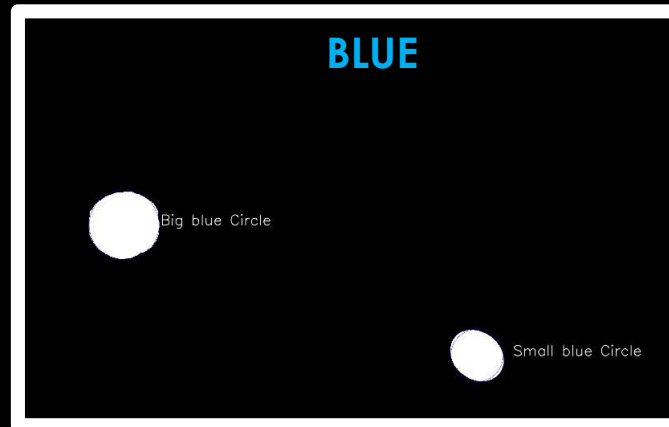
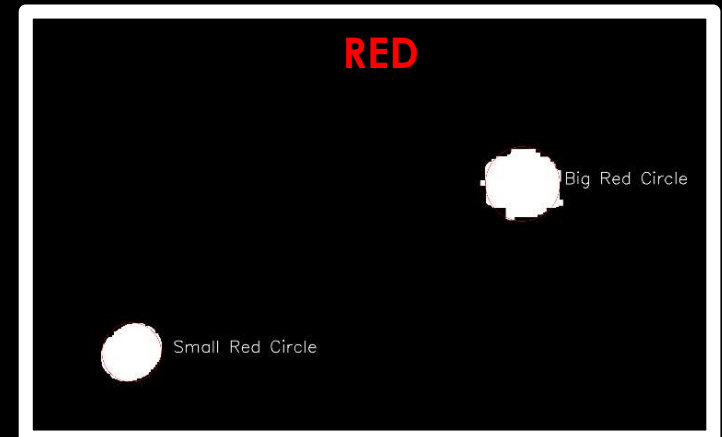
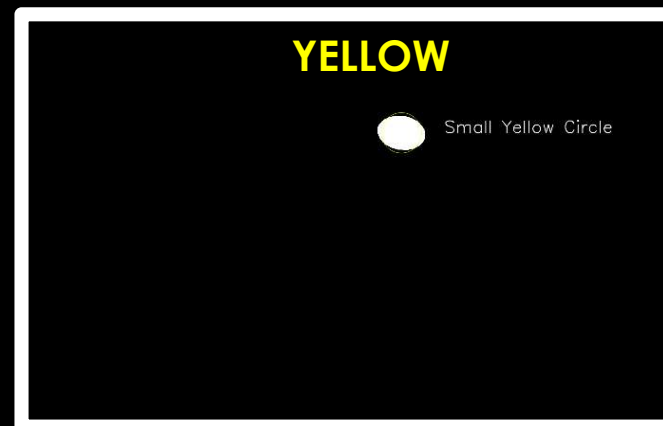
How we get the centroid and size from a blob

- #Label the red blobs found
- for `keypoint` in `keypoints`:
- **`x = keypoint.pt[0]`**
- **`y = keypoint.pt[1]`**
- **`s = keypoint.size`**
- `cv2.putText(red_mask, 'Big Red Circle', (x,y), cv2.FONT_HERSHEY_SIMPLEX, 2, (255, 255, 255), 2, cv2.LINE_AA)`

- Find the coordinates of the 4 sticker dots in test1.jpg Using Thesholds, morphological image processing and blob detection
- Use the put Text command to label what the blob represents



TASK FOR WORKSHOP:



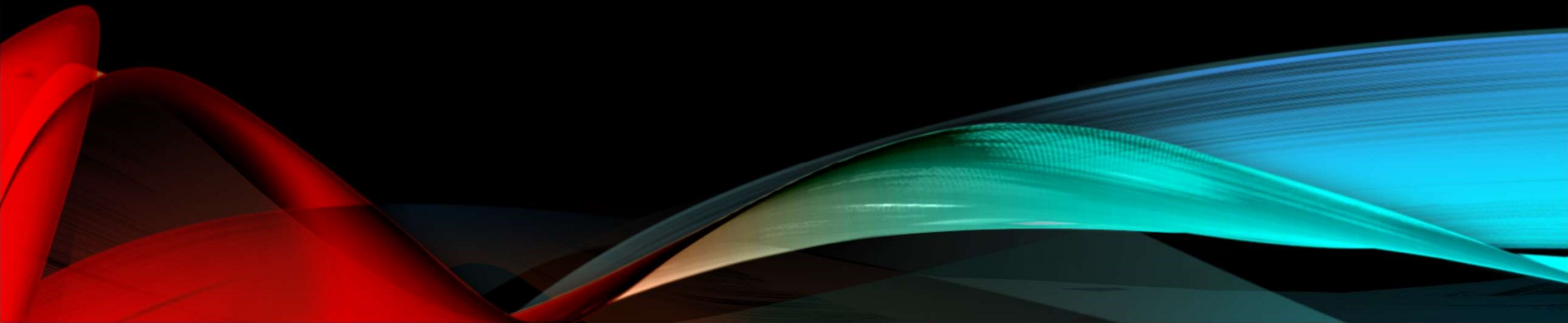


NEXT WEEK'S OBJECTIVES (LAST OPENCV WORKSHOP IN LAB)

- Locating the Position of the balls using Perspective Correction by applying Homography
- Maybe some other topic you guys suggest...
- After Next week and the student break, we will turn the code from these workshops into a 'vision system' on the Pi, sampling captured frames from the Pi camera instead of the test images here.

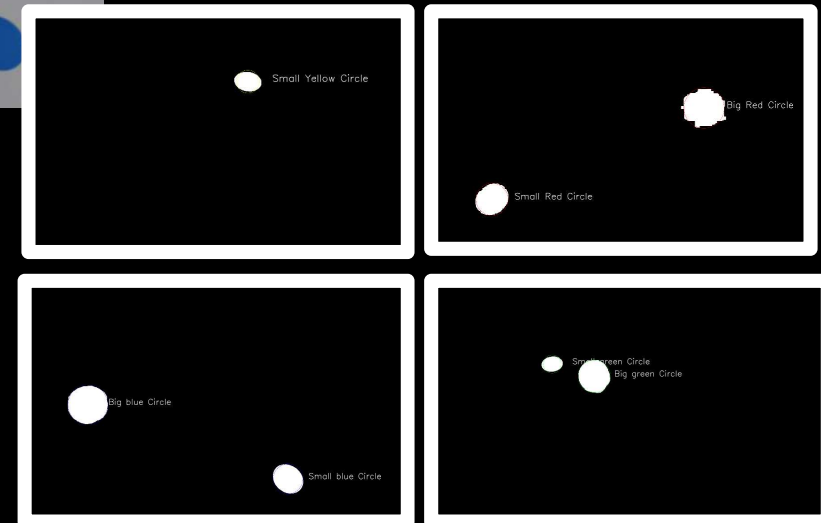
IMAGES TO ACTION

Methods to take image features to Robot actions



CONTINUING FROM LAST WEEK

- We used morphological image processing to fix thresholding defects.
- We used blob detection to extract features such as the centroid of the stickers in the image.
- The Problem is that now we want to figure out how to use that data as an input for robot actions our robot



ROBOT ARM PICKING PROBLEM

- We want to turn the image features we detected previously into coordinates for picking up the balls.
- There are a mainly 2 methods we can apply:
- Reactive –at every iteration, the robot sees and moves. (usually better in dynamic scenarios can be simple or complex)
- Planning – Determine a path for the robot to go to based on the image data collected. (does a lot of computation initially and then executes the whole action this can be easier to deal with sometimes)



REACTIVE EXAMPLE: POTENTIAL FIELD IN SOCCERBOTS

- In Mechatronics Design Project 2, you guys would do a soccerbot project.
- What potential field does is that using the detected objects, you make a certain direction more attractive or repulsive than other directions.
- E.g. the orange ball has high attractiveness and the black obstacle has repulsiveness. When the goal is in your possession, the yellow or blue goal has high attraction
- The robot will turn towards the area with high attractiveness and move in that direction until the next state in execution



- Generate obstacle map from perception, goal map from desired action
- Subtract the obstacle map from the goal map to get the motor heading map
- The peak is the direction to travel in



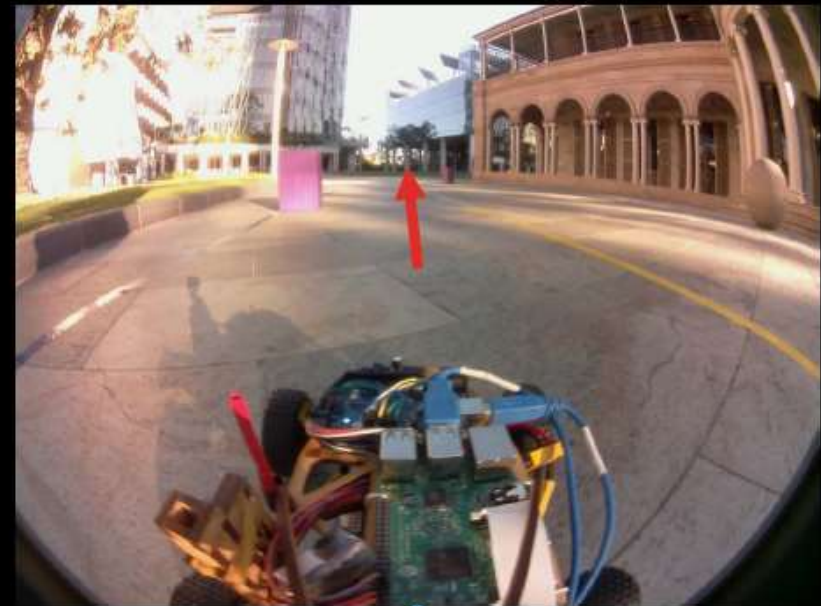
Browning, 2000



$$MHM_i = GD_i - OM_i$$

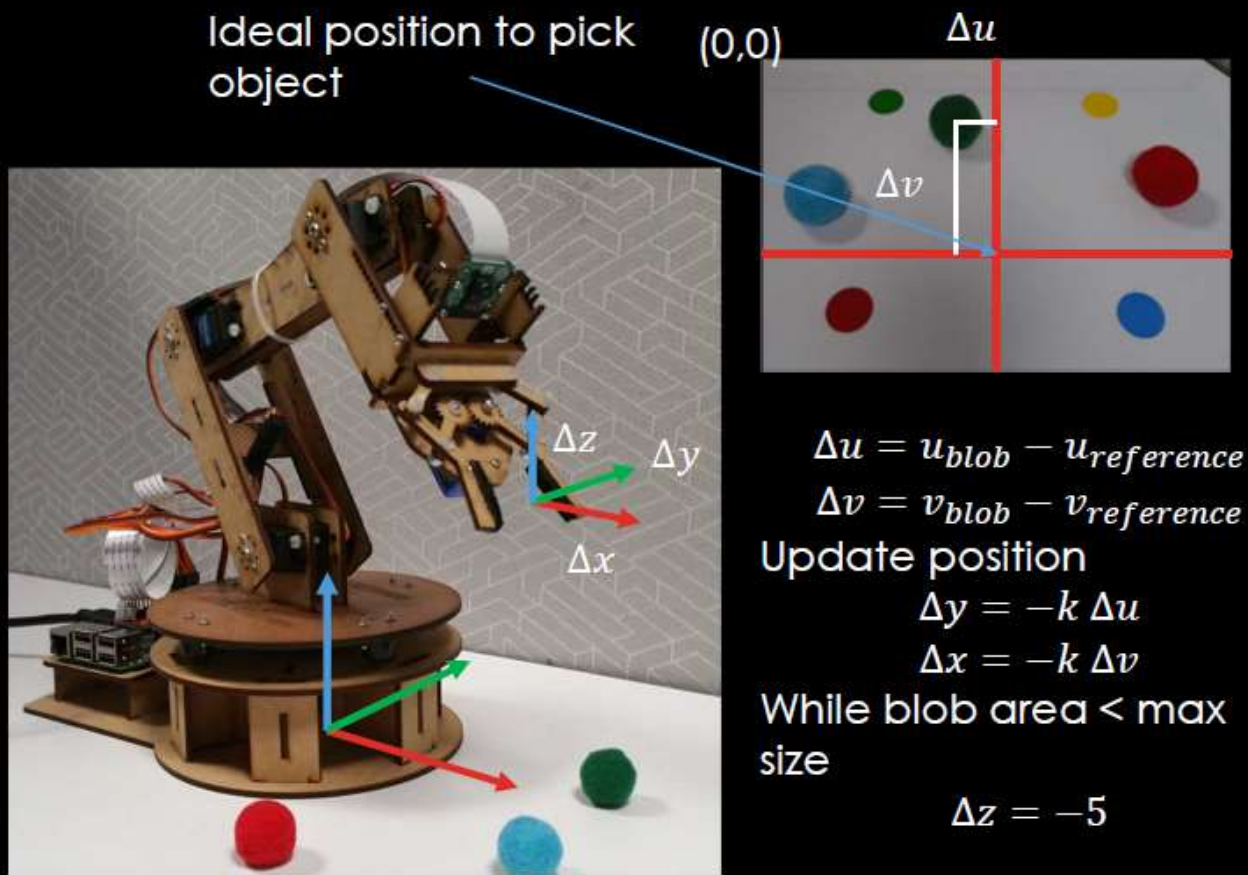
REACTIVE EXAMPLE IN DROID RACING CHALLENGE

- This can be used in DRC too!
- The droids is repulsed from the yellow and blue lanes, the purple obstacle and the other red vehicles (or attraction if you want to knock out the competition he he he...)
- Its all a matter of how you control the gains for robot motion that's steering and velocity for a mobile robot
- The robot steers until it gets the desired direction i.e. the most attractive path is in the middle pixel coordinate ->



REACTIVE CONTROL FOR ROBOT ARM

- For a robot arm, you can do a similar thing. The robot can face the scene and measure the error in image feature position ($\Delta u, \Delta v$) with reference to the position where the gripper can pick it up.
- The error goes as a gain to increment the position using updates of $\Delta x, \Delta y, \Delta z$.
- You can also use area of the blob as a limit to how much Δz you do
- You just need to know how image feature error changes as the camera moves so that the error converges
- This is easier to think of it in a simple trajectory (like moving downwards). This is basic 'visual servoing' but it is a lot trickier when you need to rotate the camera too...



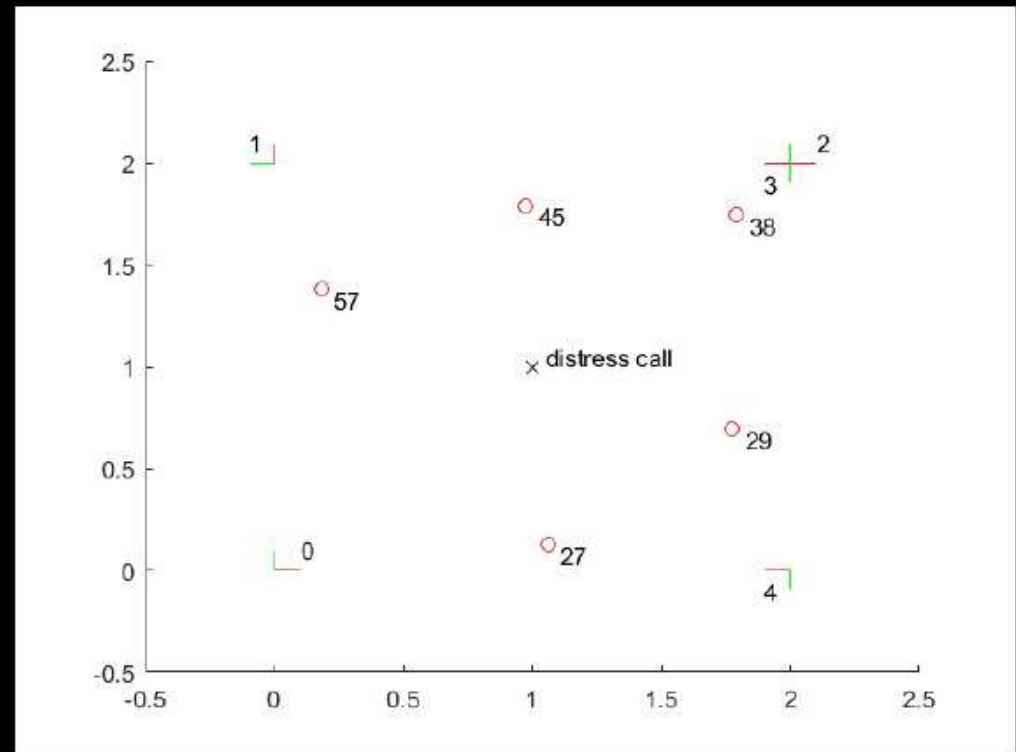


PROS AND CONS OF REACTIVE MODEL

- Pros:
 - Simple control system that can be fast
 - Has Feedback and Reacts to dynamic changes in the environment
- Cons
 - Does image processing each step so the speed is at the mercy of the computation time for each iteration.
 - Be careful what happens in-between iterations it could be that it misses something and gets stuck.
 - Sometimes hard to formulate a controller for the problem
 - Attraction and repulsion can cancel each other out making your robot stuck

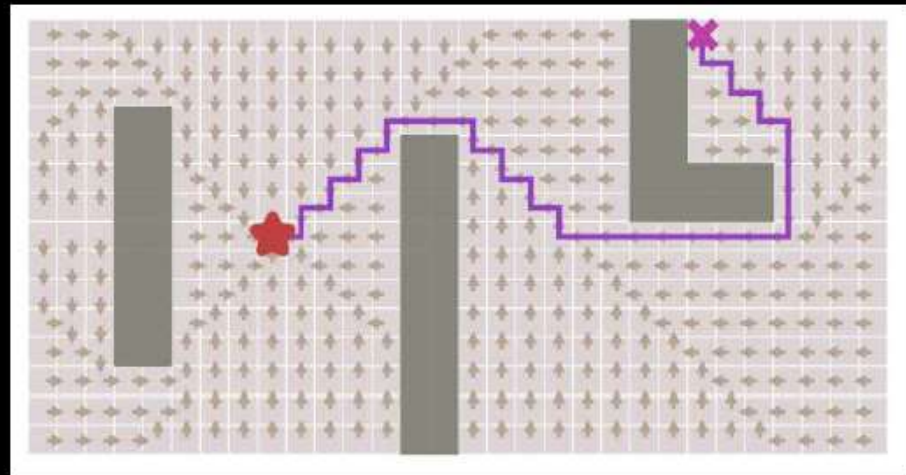
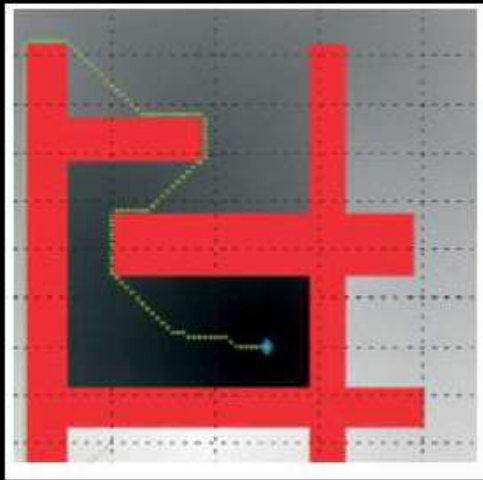
PLANNING EXAMPLE: GENERATING A MAP AND USING IT

- Sometimes it may be easier to plan the trajectory rather than reacting to it. Lets say we recorded a map of the environment (like recording the maze paths after some initial line following) we can then plan a motion towards the goal.
- E.g. in Mech Project 1: the line follower, you can record the path using a reactive method and then execute a fast planned trajectory afterwards
- E.g. in Advanced Robotics: We generate a map using SLAM (simultaneous Localisation and mapping). We then plan how to move to the place to send rescue supplies



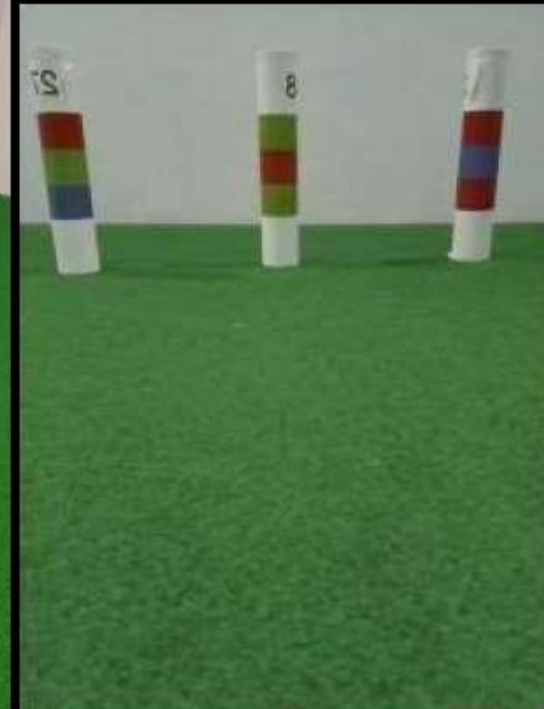
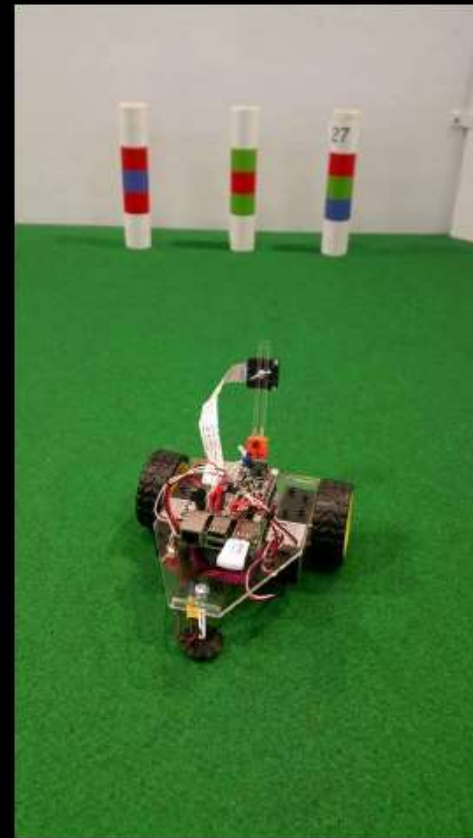
PLANNING EXAMPLE: GRAPH SEARCH

- In a maze you can use an AI search algorithm to move to a target around some obstacles to reach the end. But this is getting off topic...
- This is some AI applications which involve path planning. You may find A* a popular search algorithm find out more here:
<https://www.redblobgames.com/pathfinding/a-star/introduction.html>



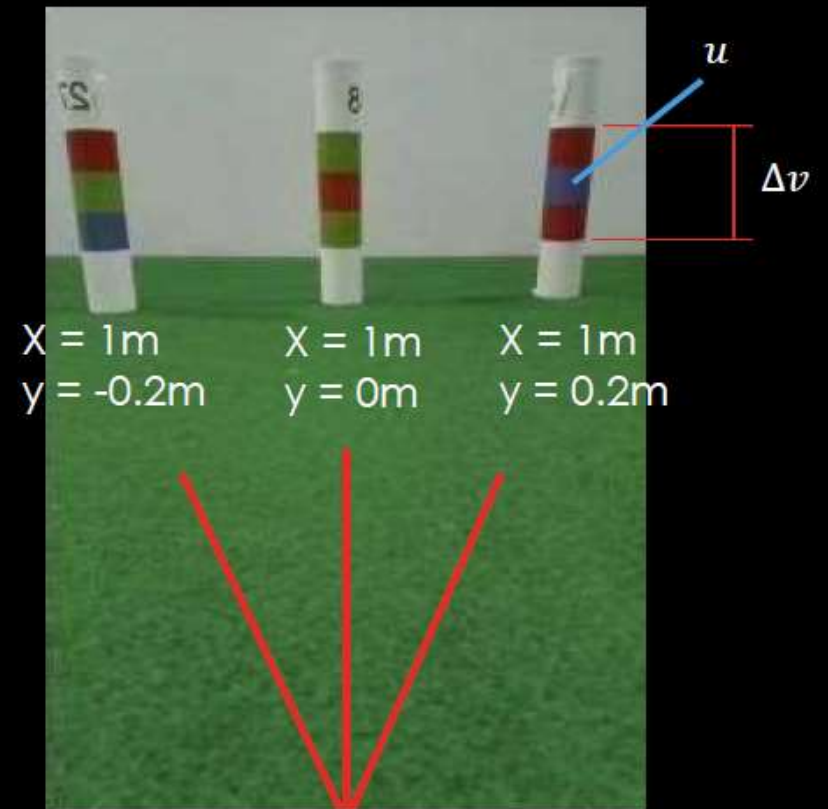
PLANNING EXAMPLE: LOCATING THE BEACONS TO MAKE THE MAP

- What's the main challenge in planning? That's probably locating where everything is accurately
- In Advanced Robotics you need to estimate where the beacons are from the camera i.e. range and bearing, to put onto a map your building
- There is no formal method given in the subject but there are a few simple methods people think of or are suggested by tutors in the subject.
- A simple method is to map an function that relates the distance to the object with something like the area of the blob.



PLANNING EXAMPLE: LOCATING THE BEACONS TO MAKE THE MAP

- When I did this subject I made an exponential regression to get the function that maps the height of the beacon colours to the distance it actually was. This involves taking some measurements of specifically placed targets
- You can have a similar function for bearing angle and u position:
- $Bearing = m * u + c$
- $X = e^{-k\Delta v} + c$
- $Range = \frac{X}{\cos(Bearing)}$





PROS AND CONS IN PLANNING

- **Pros:**

- Does motion swiftly by putting all the hard computations in the planning part
- Can be easier to design a state machine program with it as it separates the parts
- Can plan optimal paths

- **Cons:**

- Realises on the approximation of mapping/locating objects in the environment. Poor estimation gives 'open loop' fails like missing the target completely
- Fails in Unstructured and dynamic environments unless prior assumptions are done



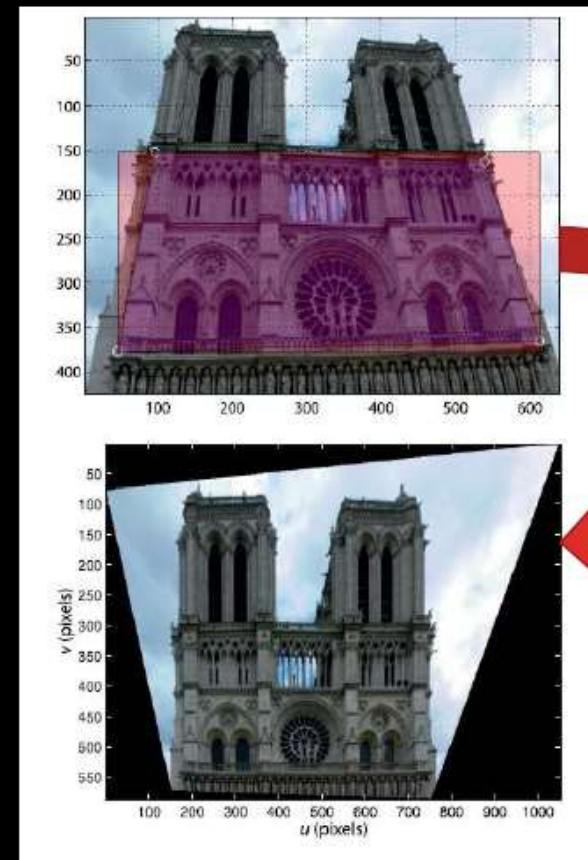
PLANNING IN THE ROBOT ARM

- We can use the camera to estimate where the objects are in the robot's reference frame.
- This may involve determining where the object is in reference to the camera or to predefined objects in the robot space.
- For our task we will be given the position of the four stickers and we will determine where the objects are with reference to them.
- To complete this we will use homography

HOMOGRAPHY: FROM TABLE TO TASK SPACE

- Homography is a transformation from one image or plane to another. With it we can do things like make that cathedral looking face on rather than at that angle
- To do this we need at least 4 points that relate the two images to solve the transformation matrix. Note the 3rd value is a scale of the x,y coordinate which should be normalised to 1

$$\begin{bmatrix} x_1 \\ y_1 \\ 1 \end{bmatrix} = H \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix} = \begin{bmatrix} h_{00} & h_{01} & h_{02} \\ h_{10} & h_{11} & h_{12} \\ h_{20} & h_{21} & h_{22} \end{bmatrix} \begin{bmatrix} x_2 \\ y_2 \\ 1 \end{bmatrix}$$

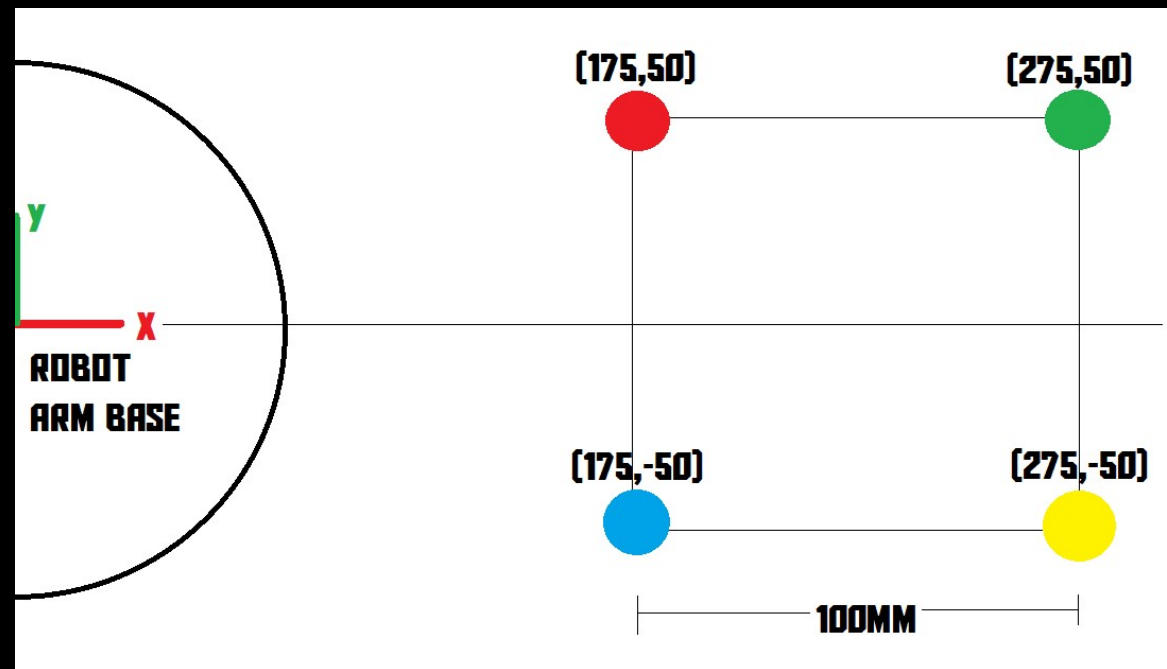


THIS IS DONE IN INTRO TO ROBOTICS BUT IN MATLAB

- Code in OPENCV Python
- #The 4 small points from source image
- `pts_src = np.array([red_dot, yellow_dot, green_dot, blue_dot])`
- #Coordinates in actual cartesian space where 1 is a scale term
- `pts_dst = np.array([[175, 50, 1],[275, -50, 1],[275, 50, 1],[175, -50, 1]])` #yaxis 50,-50
- # Calculate Homography
- `h, status = cv2.findHomography(pts_src, pts_dst)` #h is the transform from that image to that
- # Warp source image to destination based on homography
- `im_homography = cv2.warpPerspective(imageBGR, h, (300,250))`
- #Find coordinate of target by matrix multiplication:
- `pts_target = np.array([x,y,1])`
- `coordinate = np.dot(h,pts_target)`

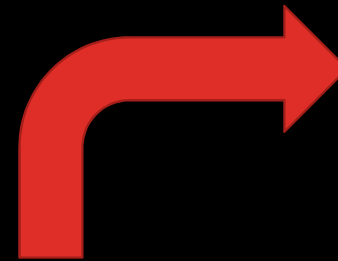
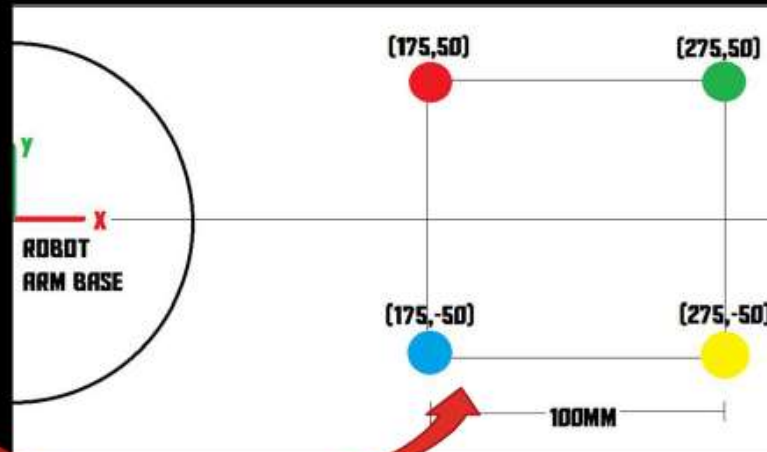
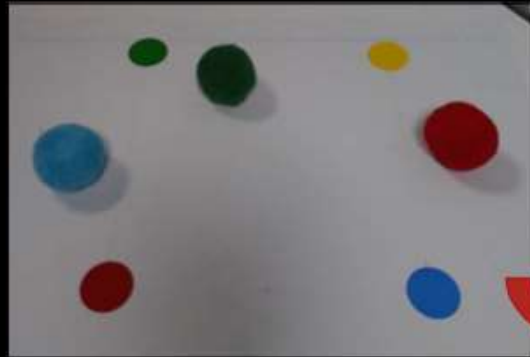
HOMOGRAPHY IN OPENCV FOR OUR APPLICATION

- We know where the 4 stickers are located at positions relative to the base of the robot arm.
- Since we have coordinates of the 4 stickers from the camera blob detection, we can find the transform and use it to locate the picking targets.
- E.g:
<https://www.learnopencv.com/homography-examples-using-opencv-python-c/>



TASK FOR WORKSHOP

- Using the centroids of the 4 stickers, determine the Homography transform to find 3D position of the location of the picking targets
- Note: the balls have a diameter of 2cm
- Hint you may need to normalise the image coordinate $(u,v,w) \rightarrow (u/w, v/w, 1)$



ADDITIONAL CONTENT FOR OPENCV WORKSHOPS

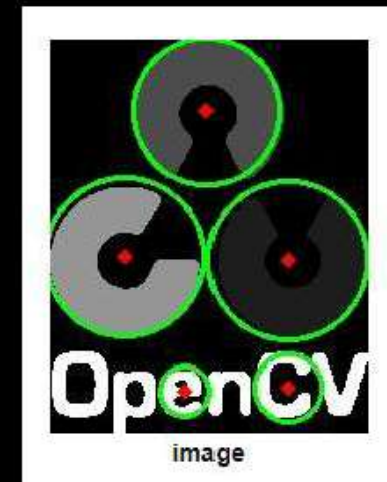
Content related to this set of OpenCV Workshops



HOUGH TRANSFORM CIRCLE DETECTION

- We could have used this algorithm instead of blobs as it can find circles in an image:

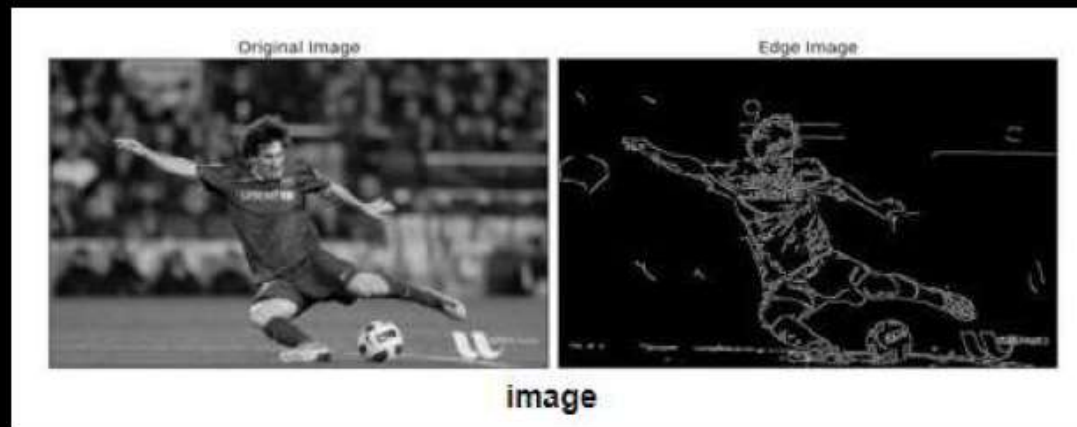
```
1 import cv2
2 import numpy as np
3
4 img = cv2.imread('opencv_logo.png',0)
5 img = cv2.medianBlur(img,5)
6 cimg = cv2.cvtColor(img,cv2.COLOR_GRAY2BGR)
7
8 circles = cv2.HoughCircles(img,cv2.HOUGH_GRADIENT,1,20,
9                             param1=50,param2=30,minRadius=0,maxRadius=0)
10
11 circles = np.uint16(np.around(circles))
12 for i in circles[0,:]:
13     # draw the outer circle
14     cv2.circle(cimg,(i[0],i[1]),i[2],(0,255,0),2)
15     # draw the center of the circle
16     cv2.circle(cimg,(i[0],i[1]),2,(0,0,255),3)
17
18 cv2.imshow('detected circles',cimg)
19 cv2.waitKey(0)
20 cv2.destroyAllWindows()
```



- https://docs.opencv.org/3.1.0/da/d53/tutorial_py_houghcircles.html

EDGE DETECTION ALGORITHMS

- There are a few algorithms to find edges including Sobel and canny:



- https://docs.opencv.org/3.1.0/da/d22/tutorial_py_canny.html
- https://docs.opencv.org/3.0-beta/doc/py_tutorials/py_imgproc/py_gradients/py_gradients.html

ROBOTIC VISION ARM PROJECT INTEGRATION



QUEENSLAND UNIVERSITY OF TECHNOLOGY
ROBOTICS CLUB

**QUTRC PICKING
CHALLENGE**

OPENCV AND RASPBERRY PI

Installing and using our OpenCV Techniques on the Raspberry Pi

PI CAMERA INSTALLATION

1. Attach Pi Camera like so to the Camera Mount:
2. Note to unlock the slot for the ribbon cable when swapping it for the longer version.
3. On the PI, go to configurations
(`$ sudo raspi-config`)
4. Go to the option to Enable the Camera, you'll need to reboot to take that into effect
5. Now you can test the quality of your image. In the terminal run this command:
`$ raspistill -v -hf -vf -o output.jpg`
6. The `-v` gives you capture information, `-hf` and `-vf` flip the image in the vertical and horizontal directions (i.e. corrects the fact that the pi camera is upside down)
7. You should have a file `output.jpg` in the PI folder, if the camera image was blank, try reconnecting the ribbon cable to ensure the connection works
8. Now install the picamera module:
`$ pip install "picamera[array]"`
9. For more details: <https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>



RASPBERRY PI OPENCV QUICK INSTALL

- You'll need internet connection... If on campus, set up a mobile hotspot and get the Raspberry Pi connected to it on if installing on campus.
- Run these commands in the terminal to install opencv quickly on the PIs
- `sudo apt-get python-opencv`
- `sudo apt-get python-matplotlib` (optional but is useful for plotting things)
- This should take some minutes and only require about 100Mb (as far as I remember)
- Once you have completed these, test the installation by opening a Python 2 IDE and running the commands:
- `import cv2`
- `cv2.__version__`

CAPTURING FRAMES FROM THE PI CAMERA IN PYTHON

- A frame is an image from a video which is a series of images. On the raspberry Pi we can take a single image or access frames from a video stream.

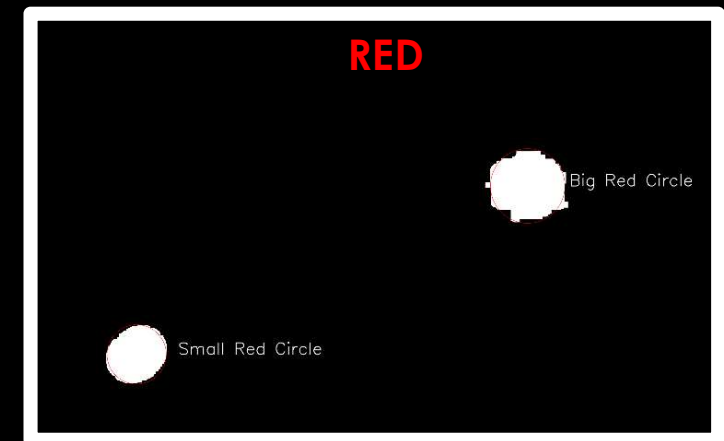
```
Accessing the Raspberry Pi Camera with OpenCV and Python
1 # import the necessary packages
2 from picamera.array import PiRGBArray
3 from picamera import PiCamera
4 import time
5 import cv2
6
7 # initialize the camera and grab a reference to the raw camera capture
8 camera = PiCamera()
9 rawCapture = PiRGBArray(camera)
10
11 # allow the camera to warmup
12 time.sleep(0.1)
13
14 # grab an image from the camera
15 camera.capture(rawCapture, format="bgr")
16 image = rawCapture.array
17
18 # display the image on screen and wait for a keypress
19 cv2.imshow("Image", image)
20 cv2.waitKey(0)
```

```
Accessing the Raspberry Pi Camera with OpenCV and Python
1 # import the necessary packages
2 from picamera.array import PiRGBArray
3 from picamera import PiCamera
4 import time
5 import cv2
6
7 # initialize the camera and grab a reference to the raw camera capture
8 camera = PiCamera()
9 camera.resolution = (640, 480)
10 camera.framerate = 32
11 rawCapture = PiRGBArray(camera, size=(640, 480))
12
13 # allow the camera to warmup
14 time.sleep(0.1)
15
16 # capture frames from the camera
17 for frame in camera.capture_continuous(rawCapture, format="bgr", use_video_port=True):
18     # grab the raw NumPy array representing the image, then initialize the timestamp
19     # and occupied/unoccupied text
20     image = frame.array
21
22     # show the frame
23     cv2.imshow("Frame", image)
24     key = cv2.waitKey(1) & 0xFF
25
26     # clear the stream in preparation for the next frame
27     rawCapture.truncate(0)
28
29     # if the 'q' key was pressed, break from the loop
30     if key == ord("q"):
31         break
```

- <https://www.pyimagesearch.com/2015/03/30/accessing-the-raspberry-pi-camera-with-opencv-and-python/>

WORKSHOP TASK:

- Using the PI camera and all that has been installed on the PI, do a video stream from the camera and make it so that it displays the thresholding for the colour red in real time.
- Adjust the thresholding to make this robust. Make this code reusable so that you can test thresholding quality in different scenarios (this can be useful if lighting conditions change a lot)
- In addition to this, get blob detection running too and see if it can detect whether it's a small or a big circle.
- Python scripts from the previous OpenCV workshops shall be distributed by USB to the PIs at the workshop.



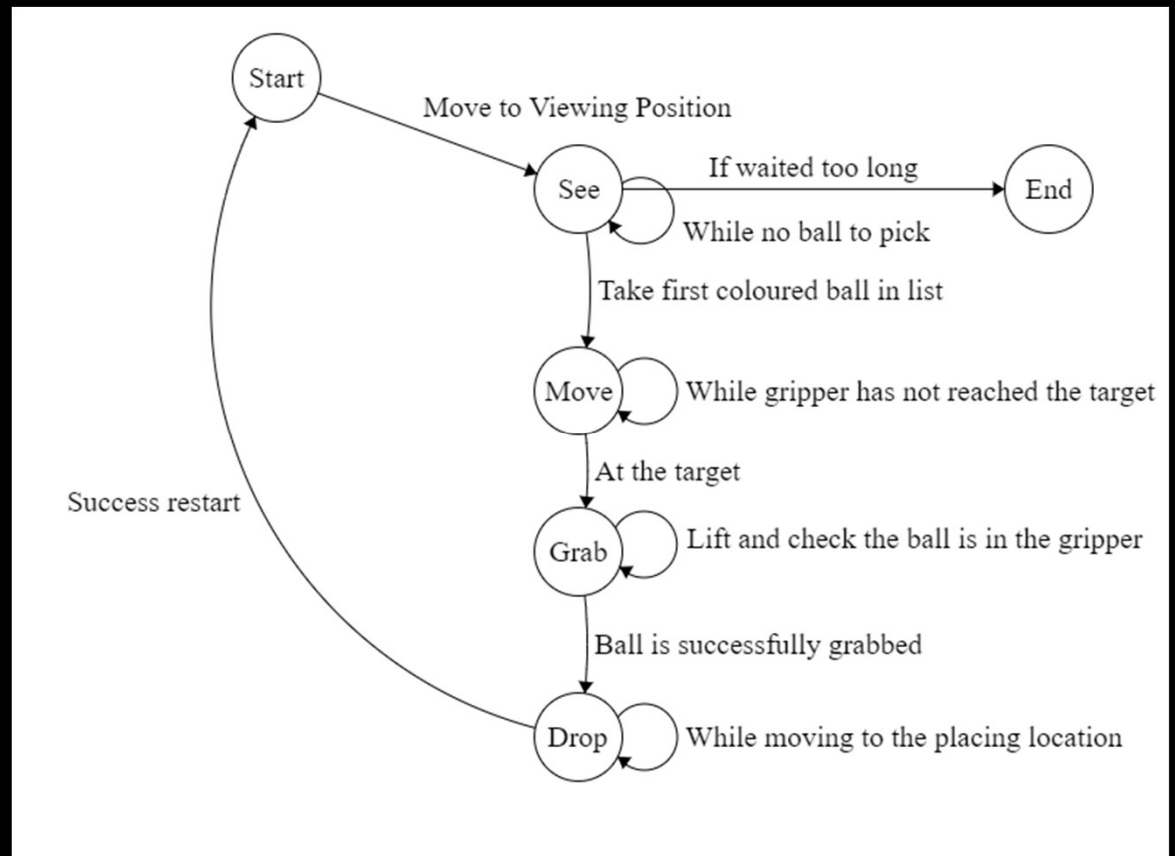


SYSTEMS INTEGRATION

- We now have a mechanical system (robot arm) and it has kinematics software
- We also have a vision system (the camera) and the software associated with that
- Now we need to use both systems to solve the automation problem which is to program the robot to see the coloured balls in the task space, pick them up and place them in a box until there are no balls for some time.
- A good structure to figure out how to program the process is to use what's called a Finite state machine. It is simply a diagram of each 'state' the robot can be in and what process is done to change from one state to the other
- Have a go at thinking about how you can structure your code for automated pick and place robot arms

FINITE STATE MACHINE

- For the automated picking task, the robot needs to see the table, move to a ball, grab it successfully, place it in the box and redo until there are no balls left and it is done waiting for any more.
- The diagram to the right describes the state machine for the problem
- State machines are simple and generally are suitable for many programming problems



TASK: COMPLETE THE PROGRAM TO GET THE ROBOT TO PICK UP OBJECTS AUTONOMOUSLY

- Tips: you can use while loops to represent each state and if statements to represent the check condition to escape the state.
- Examples of other picking machines:



END OF WORKSHOP SLIDES!



Now its just programming
and debugging!
Good Luck