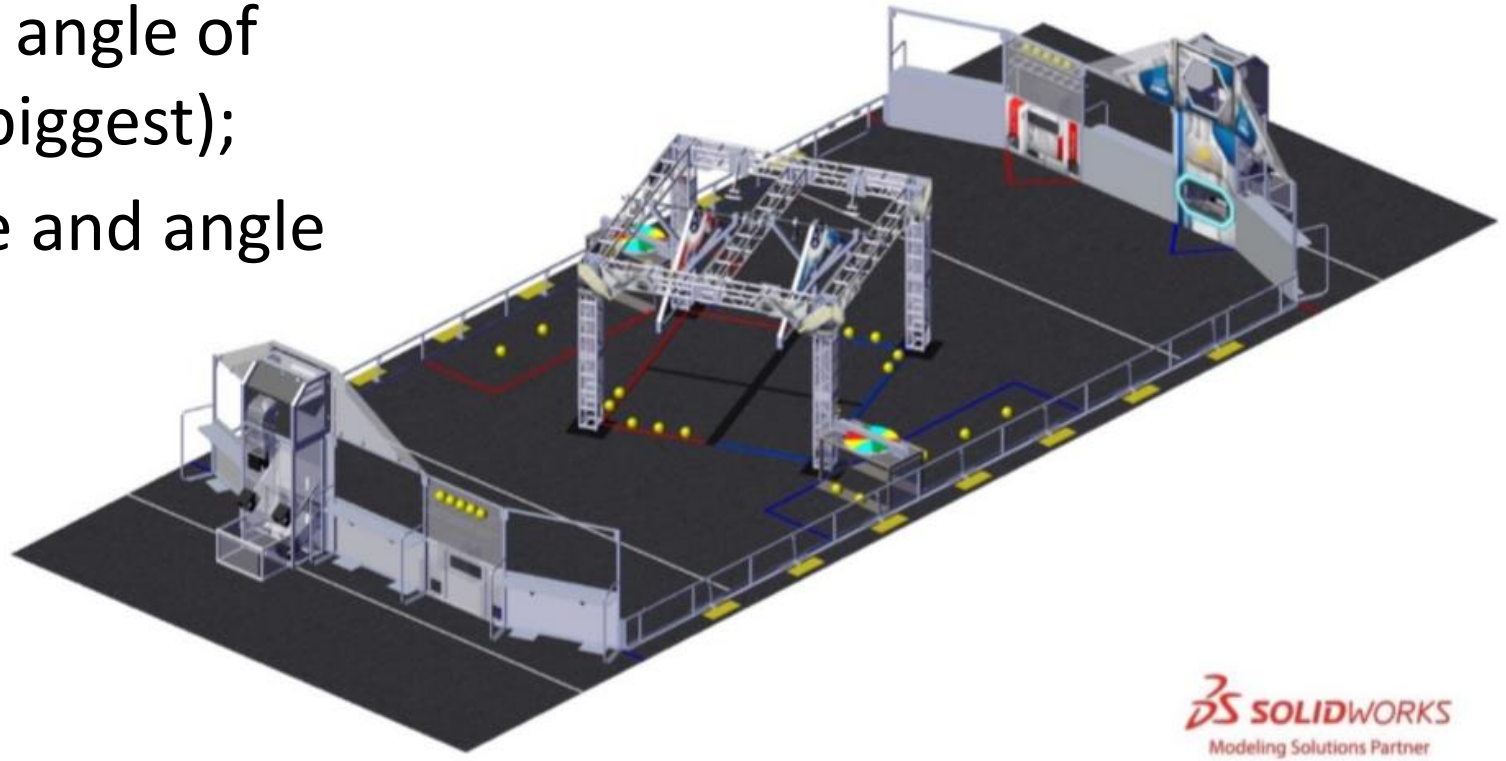
# 2020 Ball Identification

Team 1259, Paradigm Shift

# Design Requirement

- Identify yellow balls on images from camera;
- Many balls on the field and they could be overlapped on the image
- Determine the distance and angle of the ball close to the robot (biggest);
- Send the estimated distance and angle to RoboROI.



# High Level Design

Step 1 - Capture images from a camera through Raspberry pi;

Step 2 - Identify Yellow objects based on color image

Step 3 - Determine balls based on Hough Circle identification

Step 4 - Determine the location and radius of biggest yellow ball

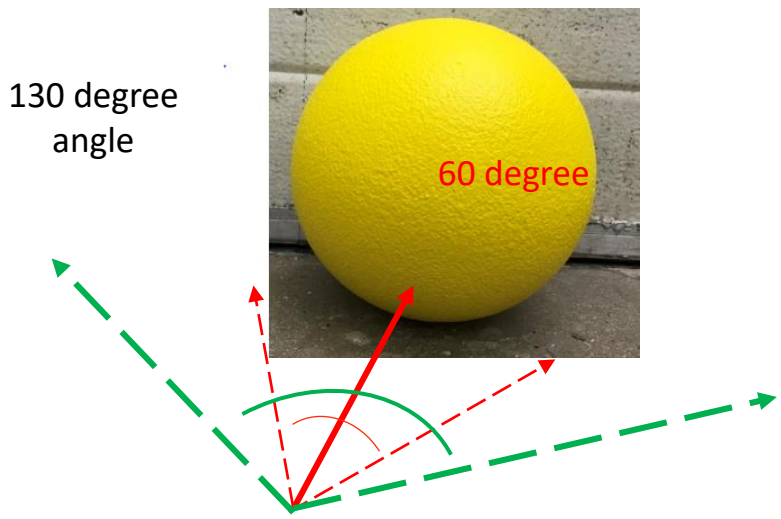
Step 5 – Estimate actual distance and angle of the ball in front of robot.

Step 6 – Send estimated information to RoboROI

Other – Error handling

# Step 1 - Capture Images from A Camera

1. Determine the type of camera after mounting location is determined.
2. Use non-distortion camera (no wide lens) to cover whole ball (7in) in an image with min distance = 6 in.




3. Determine frame rate of images to be captured by camera based on speed of robot.

Concept:

- Motion speed = X cm /sec
- Required distance, Y cm, to be updated
- Frame rate = X / Y which determines how fast camera has to acquire images per second.
- Based on frame rate and max distance to determine image resolution.

# Step 1 - Capture Images from a Camera

## Example

OpenCV function:

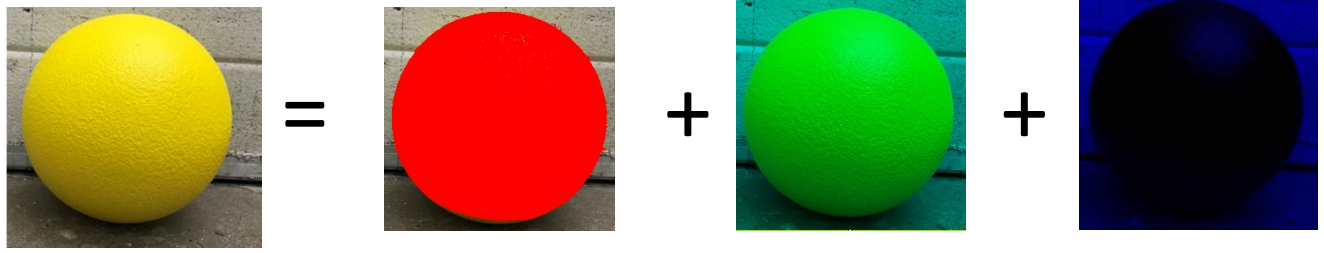
Camera.retrieve(image);

Default image size: 960x1280x3
or 480x640 where 3 indicates three images for Red, Green, and Blue color.

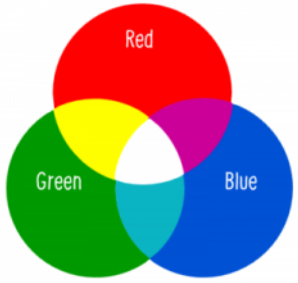
# Step 2 - Identify Yellow Ball based on Color Image

- Images from a camera have RGB (Red, Green, Blue) color.
- Usually, separating color per RGB image is not very easy due to different light environment (or different brightness);
- To make color object be easily identified and more reliably detected, HSV (Hue, Saturation, Value) color system is used. It helps identify particular color at an environment with different light intensity.
- After identifying the ball with specified color, a binary image with (0, 1) value could be used to show the ball for additional processing.

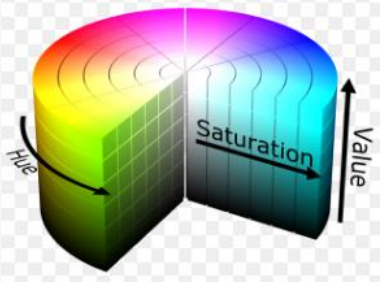
Images from camera



**RGB**




**HSV**



# Step 2 - Determine Yellow Balls based on Color

## Method ONE

```
cvtColor(image, imageHSV, COLOR_BGR2HSV);    // Convert BGR to HSV
inRange(imageHSV, lower, upper, inrange);    // Identify color per HSV image
```

Where

```
// HSV threshold to find yellow color and generate a binary image
        const Scalar lower = Scalar(22, 100, 100);
        const Scalar upper = Scalar(32, 255, 255);
```

Original

Hue

Saturation

Value

Binary image

# Step 2 - Determine Yellow Balls based on Color

## Method TWO

%% Split original image to imageRED, imageGREEN, and imageBLUE
imYELLOW = double(imageGREEN - 1.5*double(imageBLUE);

%% get max image value in imYELLOW, and then convert the image to
%% a binary image with 0 and 1 value (MATLAB code)

```
max_im = max(imYELLOW(:));
imYELLOW(imYELLOW < max_im/4 | imYELLOW > max_im ) = 0;
imYELLOW(imYELLOW > 0) = 1;
```

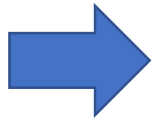
Original

imGREEN

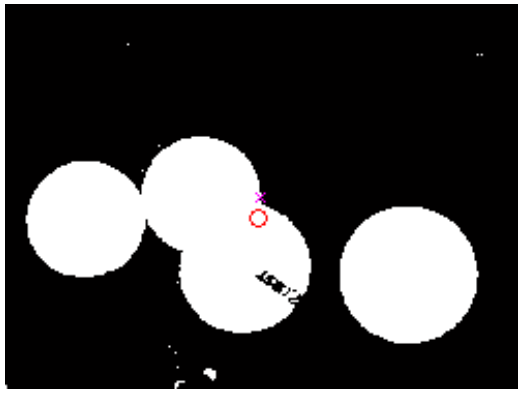
imBLUE

imYELLOW (gray color)

Threshold



Binary imYELLOW



# Step 3 - Determine Balls per Hough Circle Identification

- Circle shape on an image could be identified through Hough Transform.
- Steps:

  (1) smooth an image by a Gaussian filter to reduce the noise and minimize false circle detection

  (2) perform HoughCircles func provided by OpenCV to find edges, transform edge coordinates to (x, y) and radius per Hough transform, and then identify those circles per specified conditions.
- Detail could be found in https://docs.opencv.org/2.4/doc/tutorials/imgproc/imgtrans/hough_circle/hough_circle.html
- https://www.geeksforgeeks.org/circle-detection-using-opencv-python/

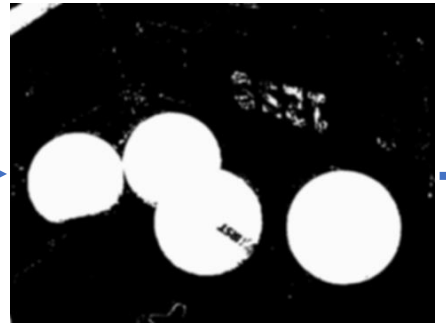
**OpenCV function in C++ ( Similar function in Python)**

```
// Perform Gaussian filter to reduce the noise and minimize false circle detection
GaussianBlur( inrange, inrange, Size(11, 11), 3, 3 );

// Apply the Hough Transform to find the circles. Settings need to be updated per camera and image size
HoughCircles( inrange, circles, CV_HOUGH_GRADIENT, 2, inrange.rows/10, 200, 100, 20, 200 );
```

Input

Filtering



Identifying circle

Green circles indicated on an original image

# Step 4 - Determine Location and Radius of a Biggest Ball

- Find a circle with the biggest radius and determine its center coordinates.

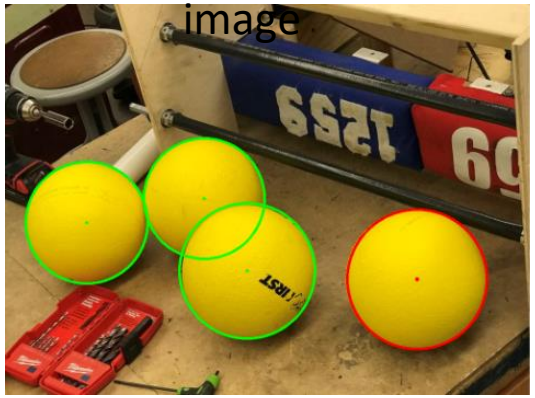
**OpenCV function in C++ ( Similar function for Python)**

```
// declear biggest_ball_radius, biggest_ball_center_x, biggest_ball_center_y
biggest_ball_radius = 0;                                f
or( size_t i = 0; i < circles.size(); i++ )
{               if (biggest_ball_radius < circles[i][2])
                {
                        biggest_ball_radius = circles[i][2];
                        biggest_ball_center_x = circles[i][0];
                        biggest_ball_center_y = circles[i][1];
                }
}
```

All circles indicated on an original image

Identifying a biggest circle

Biggest circle indicated by RED on an original image



# Step 5 - Estimate Actual Distance and Angle of the Ball - I

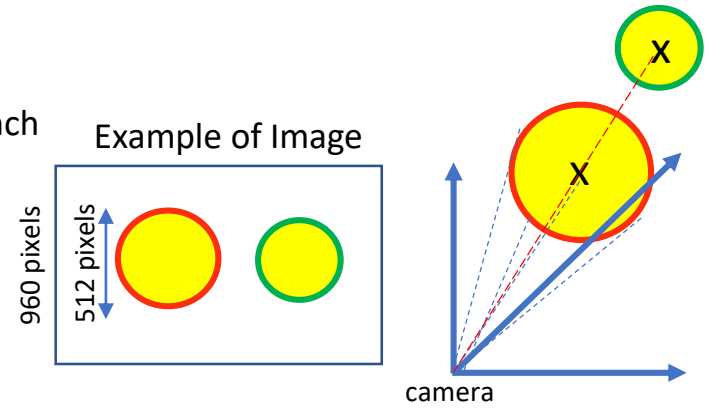
- Distance and angle of the biggest ball at different locations in front of camera could be reflected to the size of the ball and its location on the image.
- At a fixed, known distance, the size of the ball on an image could be predetermined.

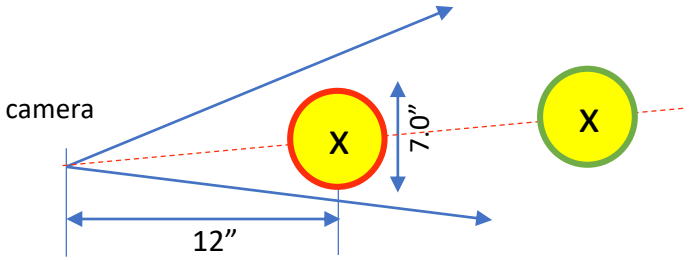
```
// Following settings is for camera calibrated values
DEFAULT_FOV_ROW_NUM          = 756;   // an example of imge from iphone
DEFAULT_HEIGHT_PIXEL         = 398;
DEFAULT_PIXEL_PER_INCH       = 72.0;  // Cube height_in_pixel / Measured Cube Height in inch
CAL_DISTANCE_INCH            = 12;
…
```

Because we could configure camera to different size to trade of resolution vs. processing speed,
So, we could scale those calibrated value per image size. That is,

```
…
standard_height_p = DEFAULT_HEIGHT_PIXEL /(DEFAULT_FOV_ROW_NUM/drawing.size().height);
pixel_per_in = DEFAULT_PIXEL_PER_INCH/(DEFAULT_FOV_ROW_NUM/drawing.size().height);
```

Example:
If image size is 640x480, different to original image size for calibration,
(1) standard height of ball in pixel is 254 (= 398 / (756/480)) at calibration distance
(2) Pixel per inch is also update, 45.7, ( = 72/ (756/480))



# Step 5 - Estimate Actual Distance and Angle of the Cube - II

- Determine the center location of the Cubic contour relative to the image center

**Total_Distance_Inch** = ((standard_height_p/(2*biggest_ball_radius))*CAL_DISTANCE_INCH);

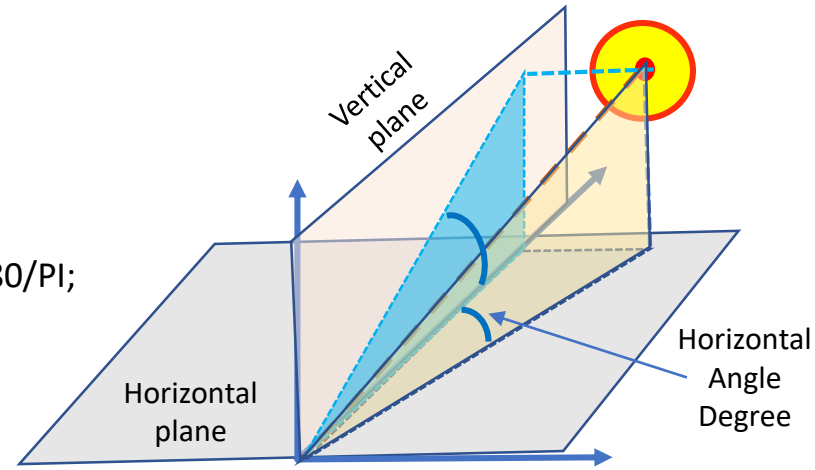
Horizontal_Distance_Pixel = biggest_ball_center_x - im_center_x;

Vertical_Distance_Pixel = im_center_y – biggest_ball_center_y;

Horizontal_Angle_Degree = atan(Horizontal_Distance_Pixel/(pixel_per_in*CAL_DISTANCE_INCH))*180/PI;

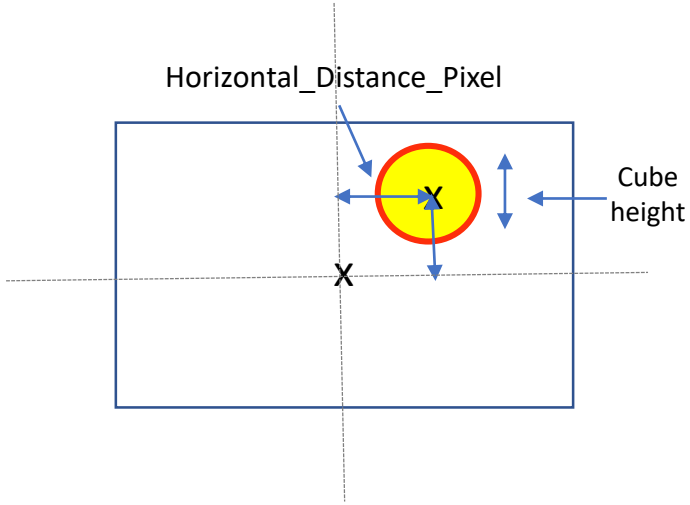
Vertical_Angle_Degree = atan(Vertical_Distance_Pixel/(pixel_per_in*CAL_DISTANCE_INCH))*180/PI;

Forward_Distance_Inch =
Total_Distance_Inch*cos(Vertical_Angle_Degree*PI/180)*cos(Horizontal_Angle_Degree*PI/180);




**“Forward Distance in inch” and “Horizontal angle in degree” are considered.**

**Another option is to send**
“Total_Distance_Inch*cos(Vertical_Angle_Degree*PI/180)”




**Note that angle and distance are updated per every image**

# Step 5 - Estimate Actual Distance and Angle of the Ball

# Example

(0, 0)

All computed parameters

// Biggest ball location and radius in pixel
Circle radius = 133
Circle center x = 793
Circle center y = 529

Center of image width = 504
Center of image height = 378
Center of the largest ball (x) = 793, relative to (0,0)
Center of the largest ball (y) = 529, relative to (0,0)
Ball height radius (pixel) = 133
Total distance (ft) = 1.49624

# Step 6 – Send Data to RoboRIO

Same as 2018 or 2019

- Initialize Network table on RoboROI
- Complete the calculation of angle and distance
- Send angle, distance, and vision counter to network table on RoboROI

- **Example of code**

```
nt_Inst = NetworkTableInstance::GetDefault();
nt_Inst.StartClientTeam(1259);
…
netTable->PutNumber("visioncounter", counter);
netTable->PutNumber("XOffAngle", Horizontal_Angle_Degree);
netTable->PutNumber("Forward_Distance_Inch", Forward_Distance_Inch);
…
```

# Other Considerations -- Error Handling

1. Most of time, camera might not get a yellow ball on the image or camera has a lens cover which will have a dark image.

   In that case, OpenCV function will not find circle or its center location

   To avoid potential error for rest computation and report, number of circles identified by Hough Transform should be checked in the code.

2. Camera needs to warm up and gets reasonable image brightness. Set 60 frames at the beginning to stabilize the brightness.

3. Add output check such as angle < 30 degree, distance < 10 feet, ball radius > 25 pixels & < image row * (1/2) to make sure that a valid result is from the image.

**NOTE: all thresholds need to be updated per camera change**