Vision Assist

Explanation:

Robot vision can be done in many ways, such as color detection, contour detection, and April Tags. Multiple vision systems can be used within the same class as well, just not at the same time. Color detection can be done to simply find the alliance, or it can be mapped to assist in driving the robot. All that implies is dividing the camera into multiple sections and comparing the level of color in each to determine the object's location. April Tags use 3rd party code to detect barcodes and the camera's position relative to the barcode.

Purpose:

Vision can serve as a sensor for many functions on the field you otherwise couldn't monitor. Scoring, parking, object avoidance. Aligning with field elements is best done through mechanical guides, but often that option is not taken. Color detection can allow a robot to perform actions with accuracy where no other sensor can.

Programming

- Scalars
- Color Conversion
- Communication

Scalars

>>Many imports are needed, these are just the ones needed for the Pipeline class. >>Colors are stored in variables called Scalars. They have 3 parameters: hue, saturation, and value. Hue is the most important as it determines where on the color wheel it is. You need a high Scalar which is a brighter denser color and a low Scalar which is dark and less visible to create a range of color to select. It is important when using online color pickers to halve the hue value as EasyOpenCv ranges from 0-> 180 instead of 0-> 255. The saturation determines where on the greyscale the color is, higher is brighter, lower is darker. Value determines the opacity. It is best to use an online color picker, such as https://colorizer.org/, to find general estimates and then fine tune the values. Make sure to test color detection in different lighting conditions as well. >>The rectangles will later be used to select target regions of the camera feed.

```
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.core.Rect;
import org.opencv.imgproc.Imgproc;
import org.opencv.imgproc.Moments;
import org.openftc.easyopencv.OpenCvCamera;
import org.openftc.easyopencv.OpenCvCameraFactory;
import org.openftc.easyopencv.OpenCvCameraRotation;
import org.openftc.easyopencv.OpenCvPipeline;
import org.openftc.easyopencv.OpenCvWebcam;

import java.util.ArrayList;

/**
 * This class intakes and analyzes the camera feed to find if the robot is aligned with the pole
 *
 * @author Zach Boeck
 * @date April 11th, 2023
 */
public class PolePipeline extends OpenCvPipeline
{
    // HSV Color Value ( Hue [0,180], Saturation [0,255], Value [0,255] )
    Scalar YELLOW_MAX = new Scalar( 20, 255, 255 );
    Scalar YELLOW_MIN = new Scalar( 10, 125, 125 );

    // Camera resolution is 600 x 448
    Rect leftRect = new Rect( new Point(230,0), new Point(430,224) );
    Rect rightRect = new Rect( new Point(430,0), new Point(630,224) );

    Mat yellow;

    private double leftAMT = 0;
    private double rightAMT = 0;

    // Minimum value for lighting/interference differences
    private final double COLOR_AMT = 1000;
```

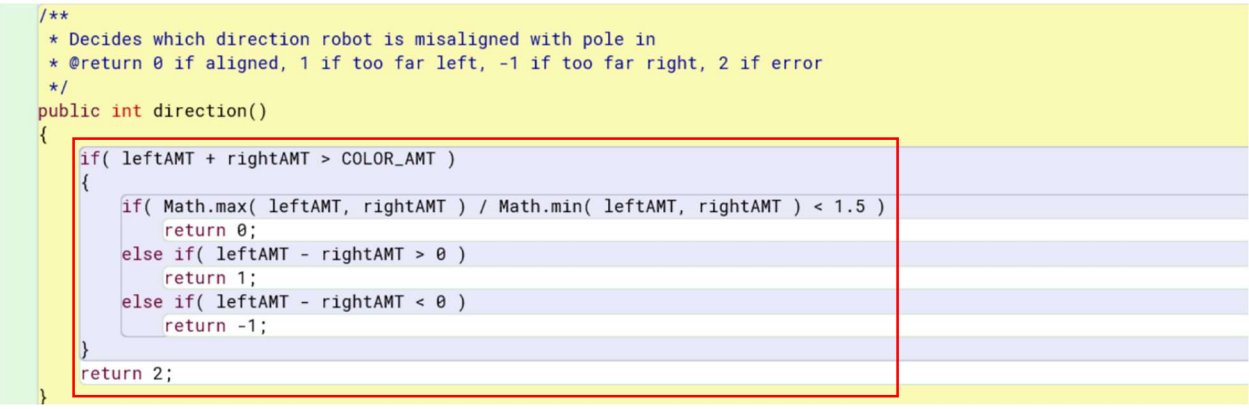
Color Conversion

>>The camera outputs frames in RGBA color space. HSV is better with vision and easier to tune so it should be converted. Although it may not be necessary, it is believed to help vision by converting from RGBA to RGB then HSV rather than just RGB to HSV. It then removes all colors not within the range between the high and low Scalars. This produces a black image with white where the selected color is. >>This image can then be divided into regions to compare with each other. A section is taken to the left and to the right of the center of the camera. It then retrieves the amount of color detected within each rectangle to compare together. >>For testing purposes, the display screen can have the rectangles drawn on the image to check the placement of them.

```
/**
 * This converts the image and extracts the amount of yellow color on the left and right side of the camera
 * @param input Input frame from camera
 * @return Image of only the yellow objects of the image
 */
@Override
public Mat processFrame( Mat input )
{
    yellow = input;

    // Converts frame from RGBA to HSV
    Imgproc.cvtColor( yellow, yellow, Imgproc.COLOR_RGBA2RGB );
    Imgproc.cvtColor( yellow, yellow, Imgproc.COLOR_RGB2HSV );

    // Removes all color values except yellow
    Core.inRange( yellow, YELLOW_MIN, YELLOW_MAX, yellow );

    Mat left = yellow;
    Mat right = yellow;

    // Crops into small target areas on left and right side of camera
    left = yellow.submat( leftRect );
    right = yellow.submat( rightRect );

    // Extracts yellow values on both sides
    leftAMT = Core.sumElems( left ).val[0];
    rightAMT = Core.sumElems( right ).val[0];

    // Adds outline to image view on Driver Station
    Imgproc.rectangle( yellow, leftRect, new Scalar(255,50,50), 5 );
    Imgproc.rectangle( yellow, rightRect, new Scalar(255,50,50), 5 );

    return yellow;
}
```

Communication

>>This method is called in autonomous to retrieve the data made by vision. It returns a single digit to tell simple rotational direction. It compares the amount of selected color from one of the target regions to the amount from the other target region. It also requires there to be enough amount of color total to prevent misinput if the camera is not pointed towards the object at all. +1,-1, and 0 are used to indicate direction and 2 indicates the object is not detected enough. The lack of detection can happen from the camera not being oriented properly, if not enough of the object is visible within the target region, or shortly after the Pipeline is started or the camera is turned on.



```
/**
 * Decides which direction robot is misaligned with pole in
 * @return 0 if aligned, 1 if too far left, -1 if too far right, 2 if error
 */
public int direction()
{
    if( leftAMT + rightAMT > COLOR_AMT )
    {
        if( Math.max( leftAMT, rightAMT ) / Math.min( leftAMT, rightAMT ) < 1.5 )
            return 0;
        else if( leftAMT - rightAMT > 0 )
            return 1;
        else if( leftAMT - rightAMT < 0 )
            return -1;
    }
    return 2;
}
```

Complete Pole Pipeline Class

Used on Noodlenose 2022-2023

To see implementation of the data:

Complete robot code can be found on the schools GitHub page

https://github.com/NicholsSchool/2023-FTC-Noodlenose

```java
import org.opencv.core.Core;
import org.opencv.core.Mat;
import org.opencv.core.MatOfPoint;
import org.opencv.core.Point;
import org.opencv.core.Scalar;
import org.opencv.core.Rect;
import org.opencv.imgproc.Imgproc;
import org.opencv.imgproc.Moments;
import org.openftc.easyopencv.OpenCvCamera;
import org.openftc.easyopencv.OpenCvCameraFactory;
import org.openftc.easyopencv.OpenCvCameraRotation;
import org.openftc.easyopencv.OpenCvPipeline;
import org.openftc.easyopencv.OpenCvWebcam;

import java.util.ArrayList;

/**
 * This class intakes and analyzes the camera feed to find if the robot is aligned with the pole
 *
 * @author Zach Boeck
 * @date April 11th, 2023
 */
public class PolePipeline extends OpenCvPipeline
{
    // HSV Color Value ( Hue [0,180], Saturation [0,255], Value [0,255] )
    Scalar YELLOW_MAX = new Scalar( 20, 255, 255 );
    Scalar YELLOW_MIN = new Scalar( 10, 125, 125 );

    // Camera resolution is 800 x 448
    Rect leftRect = new Rect( new Point(230,0), new Point(430,224) );
    Rect rightRect = new Rect( new Point(430,0), new Point(630,224) );

    Mat yellow;

    private double leftAMT = 0;
    private double rightAMT = 0;

    // Minimum value for lighting/interference differences
    private final double COLOR_AMT = 1000;

    /**
     * This converts the image and extracts the amount of yellow color on the left and right side of the camera
     * @param input Input frame from camera
     * @return Image of only the yellow objects of the image
     */
    @Override
    public Mat processFrame( Mat input )
    {
        yellow = input;

        // Converts frame from RGBA to HSV
        Imgproc.cvtColor( yellow, yellow, Imgproc.COLOR_RGBA2RGB );
        Imgproc.cvtColor( yellow, yellow, Imgproc.COLOR_RGB2HSV );

        // Removes all color values except yellow
        Core.inRange( yellow, YELLOW_MIN, YELLOW_MAX, yellow );

        Mat left = yellow;
        Mat right = yellow;

        // Crops into small target areas on left and right side of camera
        left = yellow.submat( leftRect );
        right = yellow.submat( rightRect );

        // Extracts yellow values on both sides
        leftAMT = Core.sumElems( left ).val[0];
        rightAMT = Core.sumElems( right ).val[0];

        // Adds outline to image view on Driver Station
        Imgproc.rectangle( yellow, leftRect, new Scalar(255,50,50), 5 );
        Imgproc.rectangle( yellow, rightRect, new Scalar(255,50,50), 5 );

        return yellow;
    }

    /**
     * Decides which direction robot is misaligned with pole in
     * @return 0 if aligned, 1 if too far left, -1 if too far right, 2 if error
     */
    public int direction()
    {
        if( leftAMT + rightAMT > COLOR_AMT )
        {
            if( Math.max( leftAMT, rightAMT ) / Math.min( leftAMT, rightAMT ) < 1.5 )
                return 0;
            else if( leftAMT - rightAMT > 0 )
                return 1;
            else if( leftAMT - rightAMT < 0 )
                return -1;
        }
        return 2;
    }
```