com._604robotics.robot2012.vision

Class VisionProcessing

java.lang.Object
    com._604robotics.robot2012.vision.VisionProcessing

```
public class VisionProcessing
extends java.lang.Object
```

The main class for processing camera vision on our 2012 robot. This software takes in camera images from the robot's camera, parses them, searches for pixels that look like shiny blue vision targets, blobs those pixels together, (if they are connected), and then treats it as a quadrilateral and finds the corners.

Field Summary

Fields

Modifier and Type	Field and Description
`Config`	**conf** The Configuration file for this VisionProcessing
`static VisionProcessing`	**defaultProcessing** The default VisionProcessing to use; this should be where the root of all of the vision processing is done
`VisionDisp`	**display** The display for showing the image as well as some debug data.
`static int`	**Side_Bottom** Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.
`static int`	**Side_Left** Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.
`static int`	**Side_Right** Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.
`static int`	**Side_Top** Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.

Constructor Summary

Constructors

Constructor and Description
VisionProcessing() A constructor to create a new VisionProcessing

Method Summary

Methods

Modifier and Type	Method and Description
`LinearRegression.RegressionResult`	**getRegressionForSide**(`ResultImage ri, int side, AABB guess`) Get a line that best fits the sides of a given target
`void`	**loopAndProcessPics**() This function waits for images from the image stream, processes them, and then sends results to the robot.
`void`	**loopAndProcessPreSavedPics**() This function is just a simple debug function for testing with pre-saved images.
`static void`	**main**(`java.lang.String[] args`) Just a simple main() function for running and testing the target tracking
`void`	**processImage**(`java.awt.image.BufferedImage img`) This processes the camera image and can send it to the robot (if enabled in the config file)
`static void`	**recursiveTraceBlobs**(`Img results, int i, int j, int color`)

Methods inherited from class java.lang.Object

`clone, equals, finalize, getClass, hashCode, notify, notifyAll, toString, wait, wait, wait`

Field Detail

defaultProcessing

```
public static final VisionProcessing defaultProcessing
```

The default VisionProcessing to use; this should be where the root of all of the vision processing is done

Side_Left

```
public static final int Side_Left
```

Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.

See Also:

Constant Field Values

Side_Top

```
public static final int Side_Top
```

Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.

See Also:

Constant Field Values

Side_Right

```
public static final int Side_Right
```

Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.

See Also:

Constant Field Values

Side_Bottom

```
public static final int Side_Bottom
```

Constants indicating the Left, Top, Right, and Bottom sides of a target or bounding box.

See Also:

Constant Field Values

conf

```
public Config conf
```

The Configuration file for this VisionProcessing

display

```
public final VisionDisp display
```

The display for showing the image as well as some debug data. It shows targets in green, and sides and corners in blue.

Constructor Detail

VisionProcessing

```
public VisionProcessing()
```

A constructor to create a new VisionProcessing

Method Detail

getRegressionForSide

```
public LinearRegression.RegressionResult getRegressionForSide(ResultImage ri,
                                                               int side,
                                                               AABB guess)
```

Get a line that best fits the sides of a given target

Parameters:

`ri` - the ResultImage that indicates which pixels are contained in the target

`side` - an integer indicating which of the sides to pick

`guess` - a bounding box that surrounds all of the pixels to check

Returns:

the line of best fit for the given side of the target lying in the AABB

main

```
public static void main(java.lang.String[] args)
                 throws java.lang.InterruptedException,
                        java.io.IOException
```

Just a simple main() function for running and testing the target tracking

Throws:

`java.lang.InterruptedException`

`java.io.IOException`

recursiveTraceBlobs

```
public static void recursiveTraceBlobs(Img results,
                       int i,
                       int j,
                       int color)
```

Parameters:

`results` - the Img to store returned data in

`i` - the X coordinate

`j` - the Y coordinate

`color` - the blob's color

loopAndProcessPics

```
public void loopAndProcessPics()
                        throws java.net.MalformedURLException
```

This function waits for images from the image stream, processes them, and then sends results to the robot.

Throws:

`java.net.MalformedURLException`

loopAndProcessPreSavedPics

```
public void loopAndProcessPreSavedPics()
                                 throws java.io.IOException
```

This function is just a simple debug function for testing with pre-saved images. Currently, it just reads over a loop of 50 pictures saved as target/[number].jpeg

Throws:

`java.io.IOException`

processImage

```
public void processImage(java.awt.image.BufferedImage img)
```

This processes the camera image and can send it to the robot (if enabled in the config file)

Parameters:

`img` - an image as received from the camera