VISION PROCESSING

# 2013 Vision Processing

Target Info and Retroreflection

This document describes the Vision Targets from the 2013 FRC game and the visual properties of the material making up the targets. Note that for official dimensions and drawings of all field components, please see the [Official Field Drawings](#)

High, Middle, Low Goal Targets
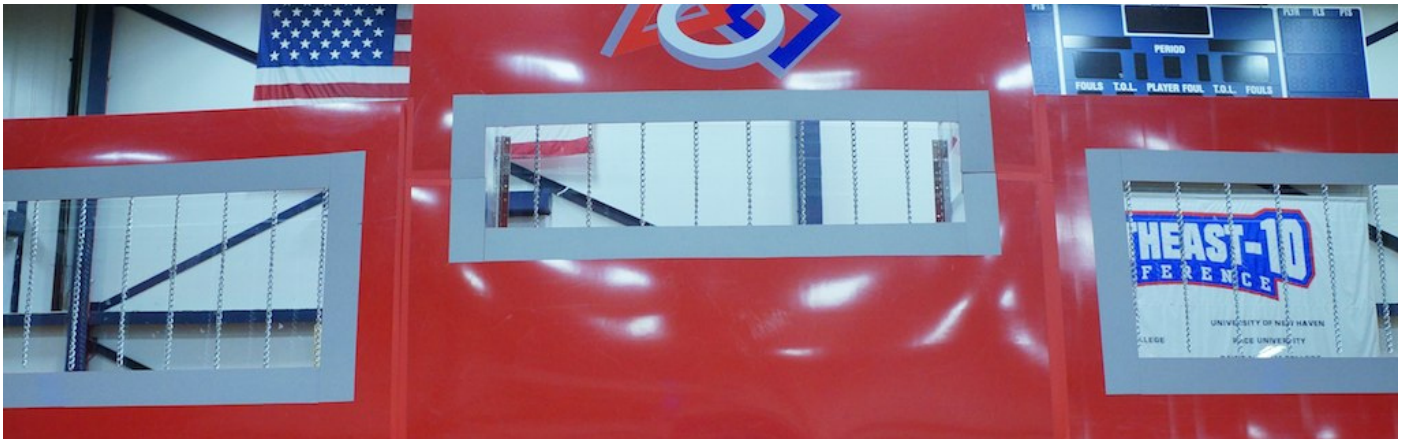
The Vision Target for the High, Middle, and Low goals consists of 4" wide retroreflective material (3M 8830 Silver Marking Film) bordering the goal opening. When properly lit, the retroreflective tape produces a bright and/or color-saturated marker.

Additional Vision Target
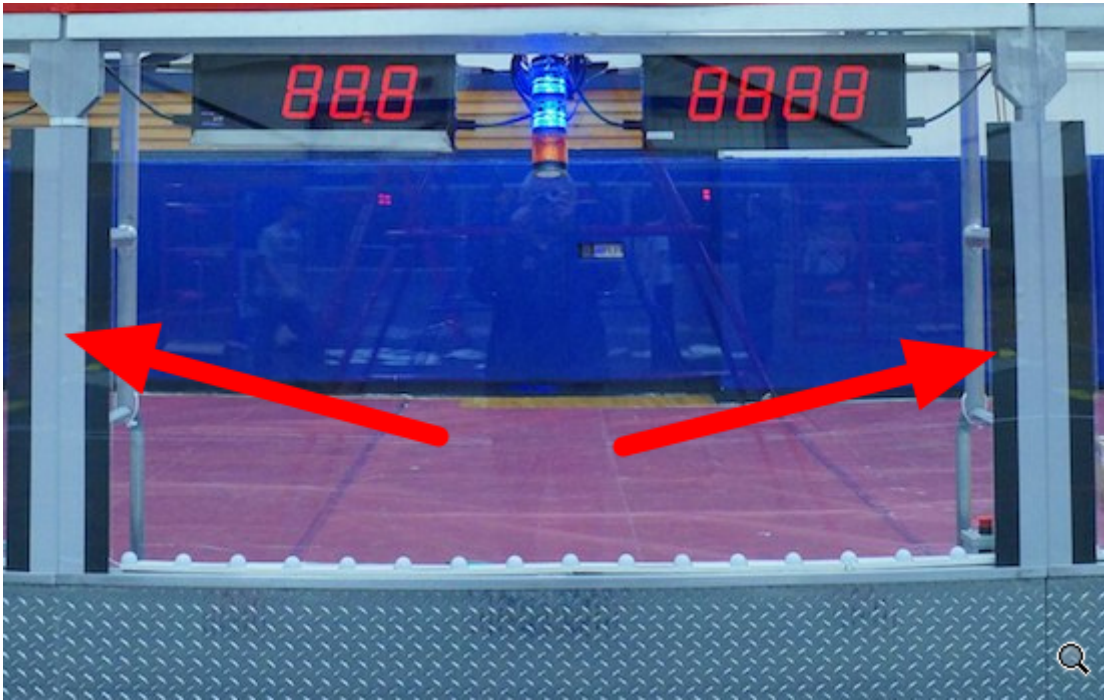
An additional Vision Target for locating the High, Middle, and Low goals is located on either side of the center player station. These targets consist of a single 33in. tall vertical stripe of retroreflective material bordered by 2in. wide black gaffers tape on the left and right sides. The stripes are located in front of the field uprights, starting at 36 in. above the carpet and spaced 72 in. apart, center-to-center.

Pyramid Goal Target

The Vision Target for the pyramid goal consists of two parts. The 20 in. tall 1-1/2 in. diameter post in the center of the pyramid goal is wrapped completely in retroreflective material. Each of the 4 horizontal metal poles which make up the exterior of the goal basket has a 4in. wide retroreflective marker centered on the pole.

Retroreflectivity vs. Reflectivity
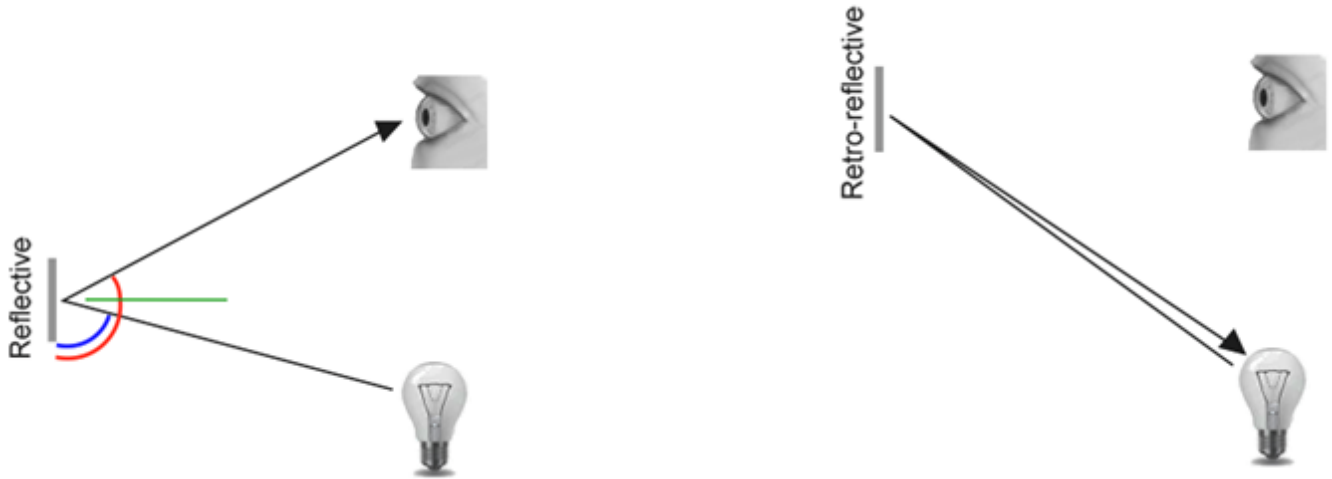
Highly reflective materials are generally mirrored so that light "bounces off" at a supplementary angle. As shown above-left, the blue and red angles sum to 180 degrees. An equivalent explanation is that the light reflects about the surface normal the green line drawn perpendicular to the surface. Notice that a light pointed at the surface will return to the light source only if the blue angle is ~90 degrees.

Retro-reflective materials are not mirrored, but it will typically have either shiny facets across the surface, or it will have a pearl-like appearance. Not all faceted or pearl-like materials are retro-reflective, however. Retro-reflective materials return the majority of light back to the light source, and they do this for a wide range of angles between the surface and the light source, not just the 90 degree case. Retro-reflective materials accomplish this using small prisms, such as found on a bicycle or roadside reflector, or by using small spheres with the appropriate index of refraction that accomplish multiple internal reflections. In nature, the eyes of some animals, including house cats, also exhibit the retro-reflective effect typically referred to as night-shine. The Wikipedia articles on retro-reflection go into more detail on how retro-reflection is accomplished.

Examples of Retroreflection
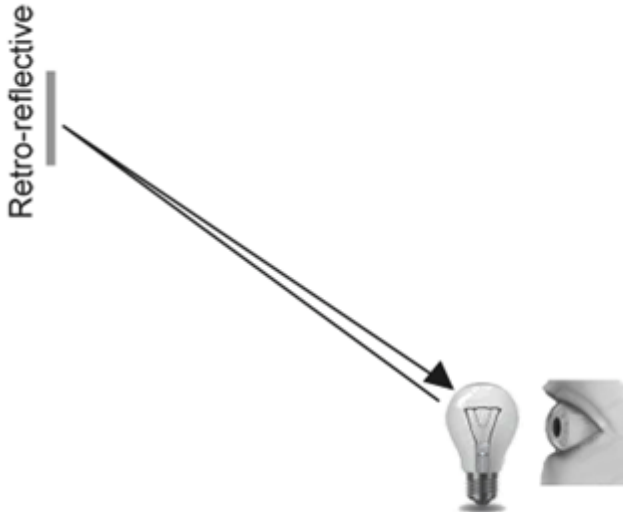
This material should be relatively familiar as it is often used to enhance nighttime visibility of road signs, bicycles, and pedestrians.

Initially, retro-reflection may not seem like a useful property for nighttime safety, but when the light and eye are near one another, as shown below, the reflected light returns to the eye, and the material shines brightly even at large distances. Due to the small angle between the driver's eyes and vehicle headlights, retro-reflective materials can greatly increase visibility of distant objects during nighttime driving.

Demonstration

To further explore retro-reflective material properties:

1. Place a piece of the material on a wall or vertical surface
2. Stand 10-20 feet away, and shine a small flashlight at the material.
3. Start with the light held at your belly button, and raise it slowly until it is between your eyes. As the light nears your eyes, the intensity of the returned light will increase rapidly.
4. Alter the angle by moving to other locations in the room and repeating. The bright reflection should occur over a wide range of viewing angles, but the angle from light source to eye is key and must be quite small.

Experiment with different light sources. The material is hundreds of times more reflective than white paint; so dim light sources will work fine. For example, a red bicycle safety light will demonstrate that the color of the light source determines the color of the reflected light. If possible, position several team members at different locations, each with their own light source. This will show that the effects are largely independent, and the material can simultaneously appear different colors to various team members. This also demonstrates that the material is largely immune to environmental lighting. The light returning to the viewer is almost entirely determined by a light source they control or one directly behind them. Using the flashlight, identify other retro-reflective articles already in your environment … on clothing, backpacks, shoes, etc.

Lighting

We have seen that the retro-reflective tape will not shine unless a light source is directed at it, and the light source must pass very near the camera lens or the observer's eyes. While there are a number of ways to accomplish this, a very useful type of light source to investigate is the ring flash, or ring light, shown to the right. It places the light source directly on or around the camera lens and provides very even lighting. Because of their bright output and small size, LEDs are particularly useful for constructing this type of device.

As shown above, inexpensive circular arrangements of LEDs are available in a variety of colors and sizes and are easy to attach to the Axis cameras. While not designed for diffuse even lighting, they work quite well for causing retro-reflective tape to shine. A small green LED ring is available through FIRST Choice. Other similar LED rings are available from the supplier, SuperBrightLEDs.com

More Information

For more information on retroreflection including types of retroreflective materials, how retroreflective performance is characterized, and information on the 3M 8830 Silver Marking Film used on the 2013 field, see the documents linked from this page.

Camera Settings

It is very difficult to achieve good image processing results without good images. With a light mounted near the camera lens, you should be able to use the provided examples, the dashboard or SmartDashboard, NI Vision Assistant or a web browser to view camera images and experiment with camera settings.

Focus

The Axis M1011 has a fixed-focus lens and no adjustment is needed. The Axis 206 camera has a black bezel that rotates to move the lens in and out. Ensure that the images you are processing are relatively sharp and focused for the distances needed on your robot.

Compression

320x240 Color Image:
Compression set to 0. Image file size is 20,715 bytes.
High quality, but large in size. May be slower to compress and decompress -- which impacts frame rate.

320x240 Color Image:
Compression set to 30. Image file size is 6,450 bytes. Good quality, relatively small in size. Some image artifacts are present on edges.

320x240 Color Image:
Compression set to 100. Image file size is 2,222 bytes. Poor quality for processing. Notice blocky artifacts and rough edges.

The Axis camera returns images in BMP, JPEG, or MJPG format. BMP images are quite large and take more time to transmit to the cRIO and laptop. Therefore the WPILib implementations typically use MJPG motion JPEG. The compression setting ranges form 0 to 100, with 0 being very high quality images with very little compression, and 100 being very low quality images with very high compression. The camera default is 30, and it is a good compromise, with few artifacts that will degrade image processing. Due to implementation details within the VxWorks memory manager, you may notice a performance benefit if you keep the image sizes consistently below 16 kBytes. **Teams are advised to consider how the compression setting on the camera affects bandwidth if performing processing on the Driver Station computer, see the FMS Whitepaper for more details.**

Resolution

Image sizes shared by the supported cameras are 160x120, 320x240, and 640x480. The M1011 has additional sizes, but they aren't built into WPILib. The largest image size has four times as many pixels that are one-fourth the size of the middle size image. The large image has sixteen times as many pixels as the small image.

The tape used on the target is 4 inches wide, and for good processing, you will want that 4 inch feature to be at least two pixels wide. Using the distance equations above, we can see that a medium size image should be fine up to the point where the field of view is around 640 inches, a little over 53 feet, which is nearly double the width of the FRC field. This occurs at around 60 feet away, longer than the length of the field. The small image size should be usable for processing to a distance of about 30 feet or a little over mid-field.

Image size also impacts the time to decode and to process. Smaller images will be roughly four times faster than the next size up. If the robot or target is moving, it is quite important to minimize image processing time since this will add to the delay between the target location and perceived location. If both robot and target are stationary, processing time is typically less important.

Color Enable

The Axis cameras typically return color images, but are capable of disabling color and returning a monochrome or grayscale image. The resulting image is a bit smaller in file size, and considerably quicker to decode. If processing is carried out only on the brightness or luminance of the image, and the color of the ring light is not used, this may be a useful technique for increasing the frame rate or lowering the CPU usage.

White Balance

If the color of the light shine is being used to identify the marker, be sure to control the camera settings that affect the image coloring. The most important setting is white balance. It controls how the camera blends the component colors of the sensor in order to produce an image that matches the color processing of the human brain. The camera has five or six named presets, an auto setting that constantly adapts to the environment, and a hold setting -- for custom calibration.

The easiest approach is to use a named preset, one that maintains the saturation of the target and doesn't introduce problems by tinting neutral objects with the color of the light source.

To custom-calibrate the white balance, place a known neutral object in front of the camera. A sheet of white paper is a reasonable object to start with. Set the white balance setting to auto, wait for the camera to update its filters (ten seconds or so), and switch the white balance to hold.

Exposure

The brightness or exposure of the image also has an impact on the colors being reported. The issue is that as overall brightness increases, color saturation will start to drop. Lets look at an example to see how this occurs. A saturated red object placed in front of the camera will return an RGB measurement high in red and low in the other two e.g. (220, 20, 30). As overall white lighting increases, the RGB value increases to (240, 40, 50), then (255, 80, 90), then (255, 120, 130), and then (255, 160, 170). Once the red component is maximized, additional light can only increase the blue and green, and acts to dilute the measured color and lower the saturation. If the point is to identify the red object, it is useful to adjust the exposure to avoid diluting your principle color. The desired image will look somewhat dark except for the colored shine.

There are two approaches to control camera exposure times. One is to allow the camera to compute the exposure settings automatically, based on its sensors, and then adjust the camera's brightness setting to a small number to lower the exposure time. The brightness setting acts similar to the exposure compensation setting on SLR cameras. The other approach is to calibrate the camera to use a custom exposure setting. To do this change the exposure setting to auto, expose the camera to bright lights so that it computes a short exposure, and then change the exposure setting to hold. Both approaches will result in an overall dark image with bright saturated target colors that stand out from the background and are easier to mask.

Identifying the Targets

Once an image is captured, the next step is to identify an Vision Targets in the image. This document will walk through one approach to identifying the 2013 targets for the High, and Middle Goals. Note that the images used in this section were taken with the camera intentionally set to underexpose the images, producing very dark images with the exception of the lit targets, see the section on Camera Settings for details.

Additional Options

This document walks through the approach used by the example code provided in LabVIEW (for PC or cRIO), C++ and Java. In addition to these options teams should be aware of the following alternatives that allow for vision processing on the Driver Station PC or an on-board PC:

1. RoboRealm
2. SmartDashboard Camera Extension (programmed in Java, works with any robot language)
3. SmartCppDashboard (Community project to provide ability to program C++ vision extensions to the SmartDashboard. **Not produced or tested by FIRST or WPI**)

Original Image

The image shown below is the starting image for the example described here. The image was taken using the green ring light available in FIRST Choice. Additional sample images are provided with the vision code examples. Some of the sample images were taken with the FIRST Choice ring light, others were taken with a variety of combinations of LED ring lights of different sizes, many with one nested inside the other.
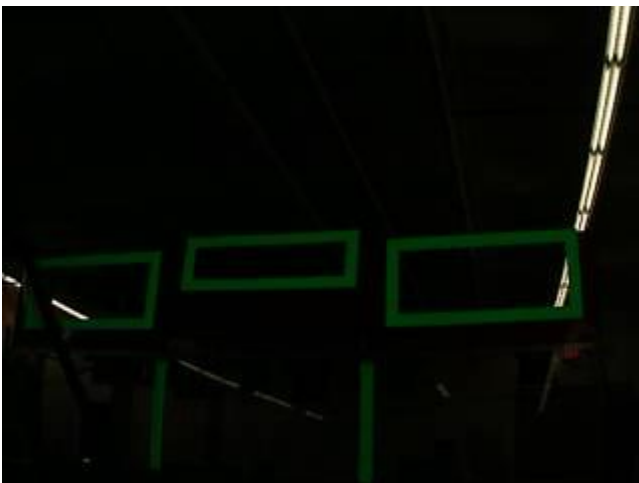
What is HSL/HSV?

The Hue or tone of the color is commonly seen on the artist's color wheel and contains the colors of the rainbow Red, Orange, Yellow, Green, Blue, Indigo, and Violet. The hue is specified using a radial angle on the wheel, but in imaging the circle typically contains only 256 units, starting with red at zero, cycling through the rainbow, and wrapping back to red at the upper end. Saturation of a color specifies amount of color, or the ratio of the hue color to a shade of gray. Higher ratio means more colorful, less gray. Zero saturation has no hue and is completely gray. Luminance or Value indicates the shade of gray that the hue is blended with. Black is 0 and white is 255.

The example code uses the HSV color space to specify the color of the arget. The primary reason is that it readily allows for using the brightness of the targets relative to the rest of the image as a filtering criteria by using the Value (HSV) or Luminance (HSL) component. Another reason to use the HSV color system is that the thresholding operation used in the example runs more efficiently on the cRIO when done in the HSV color space.

Masking

In this initial step, pixel values are compared to constant color or brightness values to create a binary mask shown below in red. This single step eliminates most of the pixels that are not part of a target's retro-reflective tape. Color based masking works well provided the color is relatively saturated, bright, and consistent. Color inequalities are generally more accurate when specified using the HSL (Hue, Saturation, and Luminance) or HSV (Hue, Saturation, and Value) color space than the RGB (Red, Green, and Blue) space. This is especially true when the color range is quite large in one or more dimension.

Teams may find it more computationally efficient, though potentially less robust, to filter based on only a single criteria such as Hue or Value/Luminance.
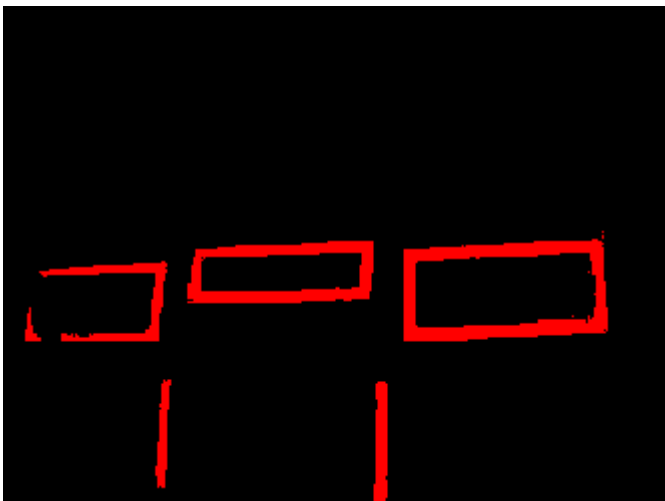
Convex Hull

The mask image contains groups of adjacent pixels referred to as particles or blobs. The next step is to identify characteristics of the target particles that distinguish it from noisy streaks as shown in the right mask, and characteristics that hold up to breaks in the rectangle caused by the pyramid, lighting variations, or other objects. It would also be preferred if the characteristics do not degrade too badly when the target is distorted as the camera moves to the side of the field and views the targets from different angles and distances.

The example code applies a convex hull operation to the mask. This operation fills interior voids and tends to repair breaks in the rectangular tape. The definition of the operation is the same as the string or rubber band rule used for robot bumpers. Each particle is surrounded by a virtual rubber band, and all pixels within the band are now added to the particle. A particle shaped like the letter V would become a downward pointing triangle; a C would become a filled-backwards D, etc. Compare the masked image above to the convex hull image below to see the results of the operation.

Particle Analysis

As shown above, the primary role of the convex hull is to fill and repair the rectangles. After the convex hull operation, a particle report operation is used to examine the area, bounding rectangle, and equivalent rectangle for the particles. These are used to compute several scored terms to help pick the shapes that are most rectangular. Each test described below generates a score (0-100) which is then compared to pre-defined score limits to decide if the particle is a target or not.

Rectangularity

The rectangle score is calculated as (Particle Area / Bounding Box Area) * 100. A perfectly rectangular particle will score 100, a circular particle will score (π/4) * 100, or about 78, and the score will drop even further for diagonal lines and other streaks.

Aspect Ratio

The aspect ratio score is based on (Particle Width / Particle Height). The width and height of the particle are determined using something called the "equivalent rectangle". The equivalent rectangle is the rectangle with side lengths x and y where 2x+2y equals the particle perimeter and x*y equals the particle area. The equivalent rectangle is used for the aspect ratio calculation as it is less affected by skewing of the rectangle than using the bounding box. When using the bounding box rectangle for aspect ratio, as the rectangle is skewed the height increases and the width decreases.

The Middle and High targets on the field have different aspect ratios, so two aspect ratio scores are generated, one for each goal type. The field target ratios are 62/29 for the Middle goal and 62/20 for the High goal. The aspect ratio score is normalized to return 100 when the ratio matches the target ratio and drops linearly as the ratio varies below or above.

Hollowness

The edge score is used to determine how hollow the rectangle is. As shown below, it is calculated using the row and column averages across the bounding box extracted from the original image and comparing that to a profile mask. The score ranges from 0 to 100 based on the number of values within the row or column averages that are between the upper and lower limit values.

1.0   0.2   0.05   0.2 1.0
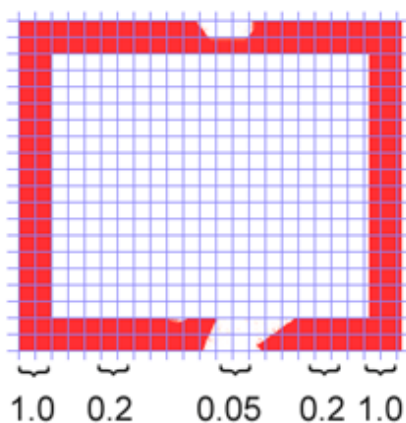
Column averages for a particle rectangle.
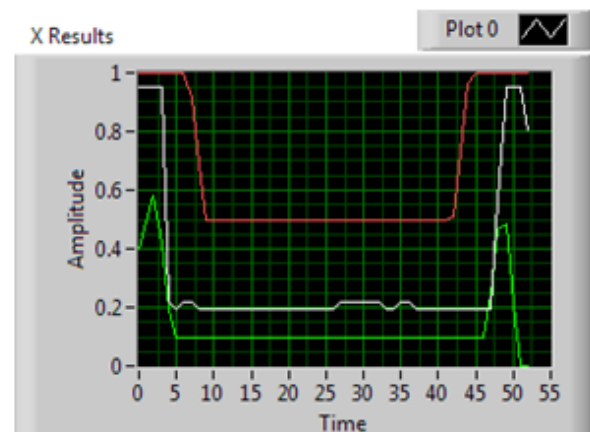
White line is the average, red is upper limit, and green lower limit.

Measurements

If a particle scores well enough to be considered a target, it makes sense to calculate some real-world measurements such as position and distance. The example code includes these basic measurements, so let's look at the math involved to better understand it.

Position

The target position is well described by both the particle and the bounding box, but all coordinates are in pixels with 0,0 being at the top left of the screen and the right and bottom edges determined by the camera resolution. This is a useful system for pixel math, but not nearly as useful for driving a robot; so let's change it to something that may be more useful.

To convert a point from the pixel system to the aiming system, we can use the formula shown below.

The resulting coordinates are close to what we want, but the Y axis is inverted, so this is corrected by multiplying the point by [1,-1]. This coordinate system is useful because it has a centered origin and the scale is similar to joystick outputs and RobotDrive inputs.

$$A_{x,y} = (P_{x,y} - \frac{resolution_{x,y}}{2}) / \frac{resolution_{x,y}}{2}$$
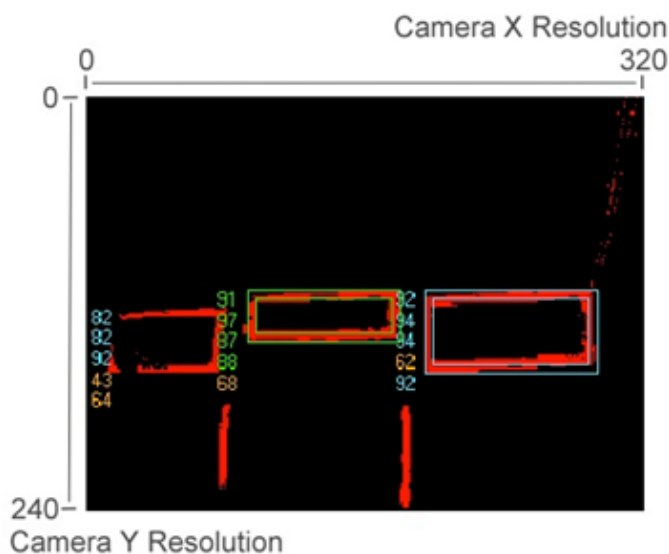
Pixel Coordinate System – $P_{x,y}$
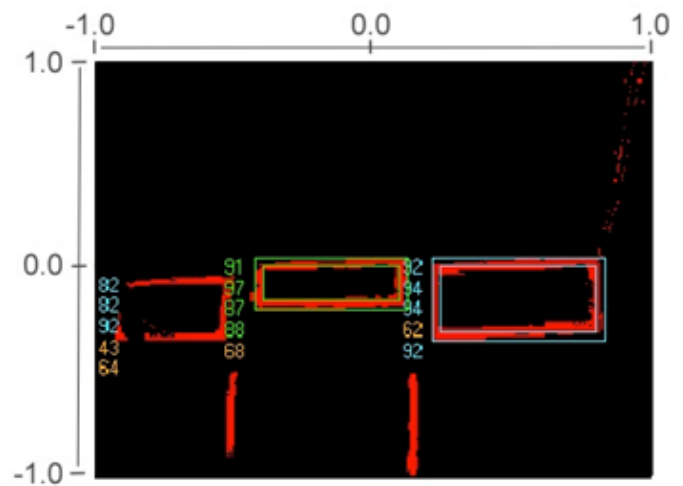
Aiming Coordinate System – $A_{x,y}$

Distance

The target distance is computed with knowledge about the target size and the camera optics. The approach uses information about the camera lens view angle and the width of the camera field of view. Shown below-left, a given camera takes in light within the blue pyramid extending from the focal point of the lens. Unless the lens is modified, the view angle is constant and equal to 2Θ. As shown to the right, the values are related through the trigonometric relationship of …

tanΘ = w/d

The datasheets for the Axis cameras can be found at the following URLs:Axis 206 and AxisM1011. These give rough view angles for the lenses of 47˚ for the M1011, and 54˚ for the 206 model. Remember that this is for entire field of view, and is therefore 2Θ.
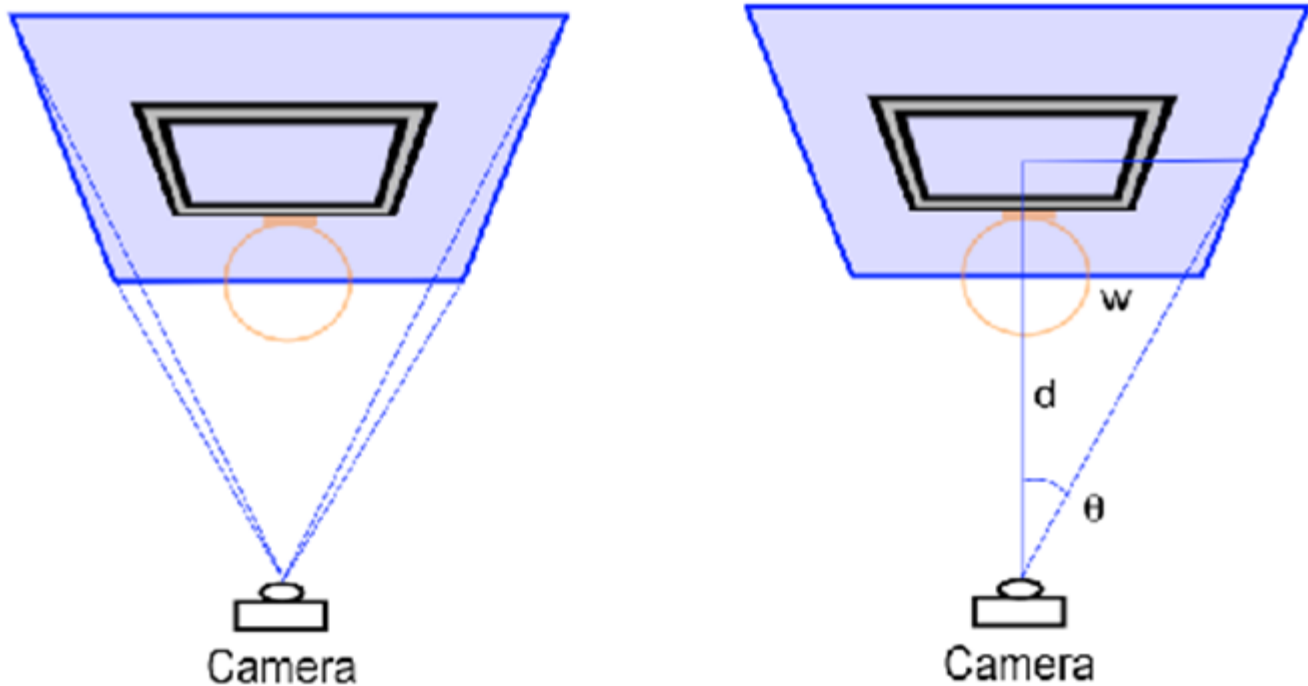
Distance Continued

The next step is to use the information we have about the target to find the width of the field of view the blue rectangle shown above. This is possible because we know the target rectangle size in both pixels and feet, and we know the FOV rectangle width in pixels. We can use the relationship of …

$T_{ft}/T_{pixel} = FOV_{ft}/FOV_{pixel}$

We also use the fact that the pixels of the camera are nearly square to calculate the field of view in the X direction despite using the target height. The X axis field of view is calculated as the view angles provided in the camera datasheets are horizontal view angles. We know that the target height measures 29 in. for the Middle goals and 20 in. for the High goals, and in the example images used earlier the High target rectangle measures 30 pixels when the camera resolution was 320x240. This means that the blue rectangle width is 20*320/30 or 213.33 in. Half of the width is 106.66 in, and the camera used was the 206, so the view angle is ~54°, making Θ be 27°. Putting this information to use, the distance to the target is equal to 106.66/tan27° or 209.33 in.

Notice that the datasheets give approximate view angle information. When testing, it was found that the calculated distance to the target tended to be a bit short. Using a tape measure to measure the distance and treating the angle as the unknown it was found that view angles of 48° for the 206 and 43.5° for the M1011 gave better results. To perform your own calibration, eliminate perspective distortion by placing the camera and target in the same horizontal and vertical plane, and use a tape measure between the camera lens and the target center point. Note the target heights in pixels for each distance, rearrange the equation to solve for the angle, and use the resulting data to determine the angle that fits the data best.

LabVIEW Code

The previous article detailed the theoretical approach to identifying the Vision Targets on the 2013 FRC Field. This article details the implementation in the LabVIEW code that matches this theoretical aprroach.

Finding the Example
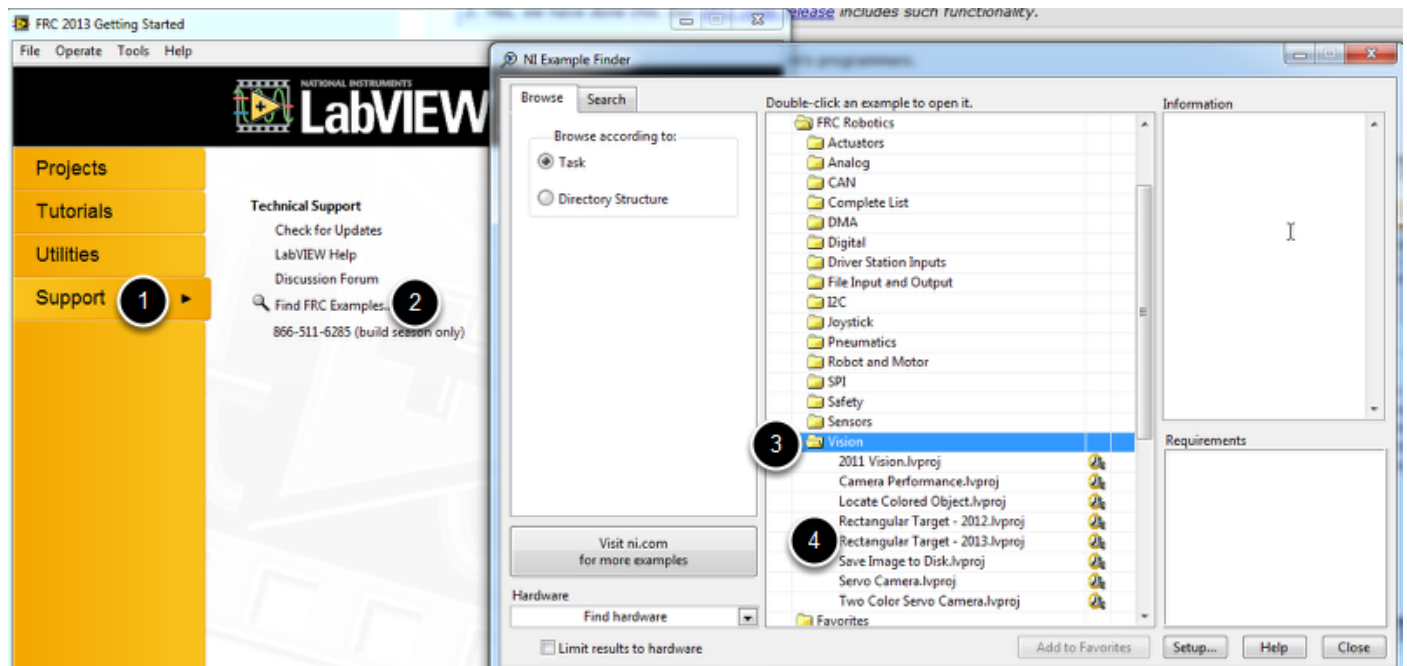
The 2013 LaVIEW example vision code is bundled along with the other LabVIEW examples. To find the example from the LabVIEW splash screen, click **Support >> Find FRC Examples** then open the **Vision Folder** and locate **Rectangular Target - 2013.lvproj**

PC or cRIO

When the example opens, you may notice that there are VIs located in both the RT CompactRIO Target and My Computer sections of the Project Explorer. Examples have been provided to run both on a PC (using a camera or stored files) as well as on the cRIO. To open the PC version double click on the **Rectangular Target - 2013.vi under the My Computer** target, to open the cRIO version, double click on **Vision Processing.vi under the RT CompactRIO Target**.

Teams should be aware that the cRIO version is a Vision Processing example only, it should integrated into your existing robot framework and not used as a starting point for writing robot code (it has no Teleop, Auto, Disabled or Test VIs).

Note that because these are 2 different VI's changes to configuration of one (such as threshold values or score limits) will not carry over to the other, you will have to mirror the changes manually if desired.

The Front Panel
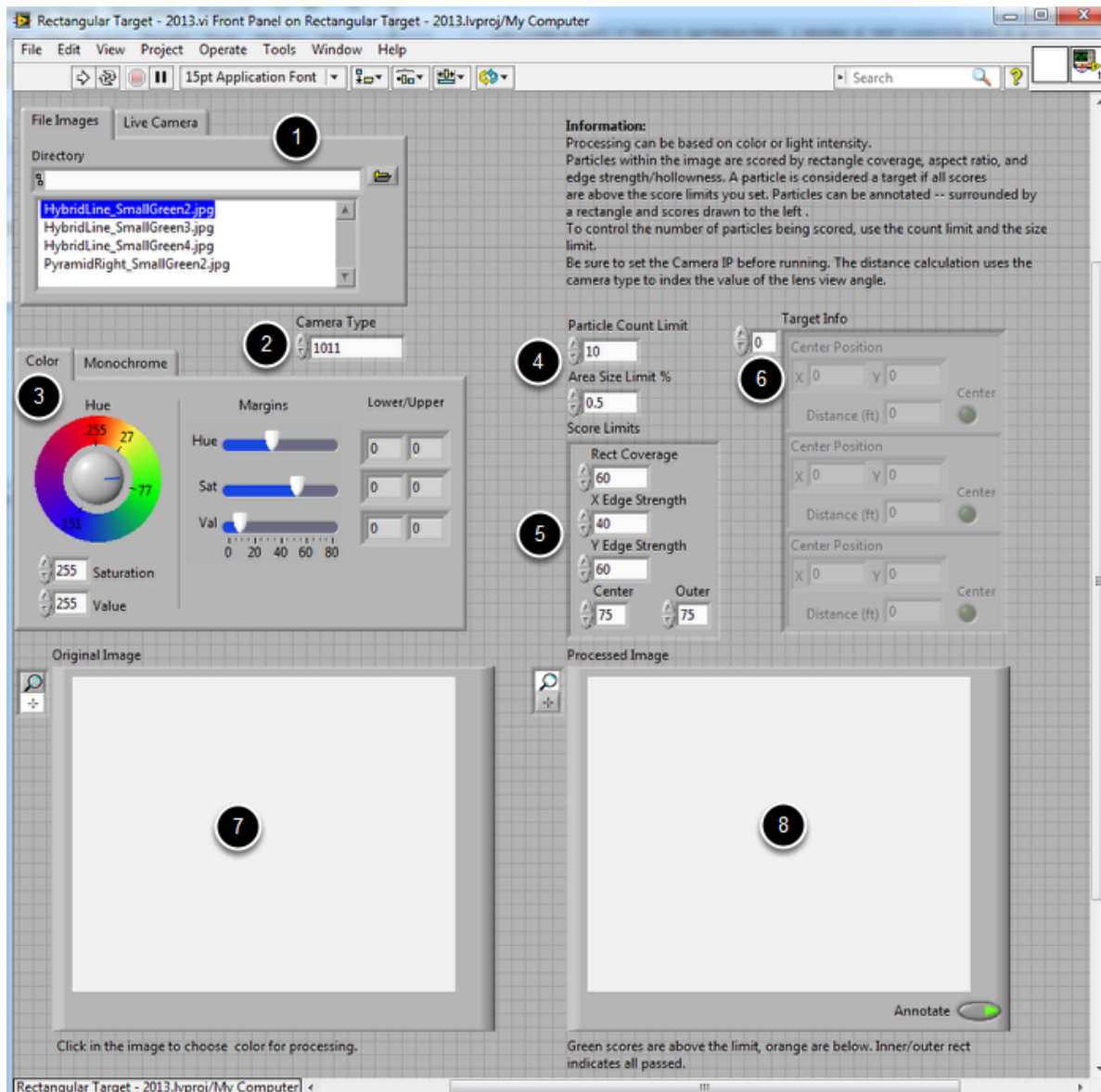
The front panel of the computer version is shown here, both versions have similar panels with the exception of the file/camera controls only appearing in the computer version. The parts of the front panel are:

1. The File/Camera controls: This section contains the controls for selecting the camera to connect to or file to process. On the File Images tab, clicking the folder icon will allow you to browse to a directory containing the desired images, then click the image in the box to select it. On the Live Camera tab you can enter the IP address of the camera to connect to and set the resolution, framerate, and compression.
2. The Camera Type: The camera type control changes the view angle used in the distance calculation based on the camera used.
3. The Color Threshold Controls: This section contains the controls used for the initial thresholding operation on the image. The Color tab has controls for the Hue, Saturation and Value and margins for each (to set the width of the range). **Note that these values can be set automatically by clicking on the desired color in the Original Image section.** The Monochrome tab contains a control to set the plane you are interested in (Hue for color, Luminance or Value for brightness) and sliders to control the desired range.

Remember that to save modified values, after the program has stopped running right click on the desired control and select "Data Operations" >> " Make Current Value Default"

4. Particle Limits: These controls limit the particles being scored by size and total number of particles.
5. Score Limits: This section sets the minimum limit for each score. A particle must exceed this minimum for each of the first three scores and one of the Aspect Ratio scores to be considered a target.
6. Target Info: This indicator contains information on identified targets if any have been detected in the image
7. Original Image: This section contains the original unprocessed image. When using Color processing, clicking anywhere in this image will set the color threshold values to detect that color.
8. Processed Image: This section contains the processed image. This is the image after thresholding, but does not show the results of the Convex Hull operation or the particle filtering. If the Annotate control is enabled, any particle that passes the particle filter will have it's score values shown next to it. Scores that exceed the minimum will be displayed in green or team, scores that are below the limit will be orange. If a particle is classified as a target, an inner and outer box will be drawn around it. Particles classified as Center targets (the High Goal) will be shown in green, particles considered outer targets (Middle Goal) will be shown in teal.

Acquiring an Image and Setting Thresholds
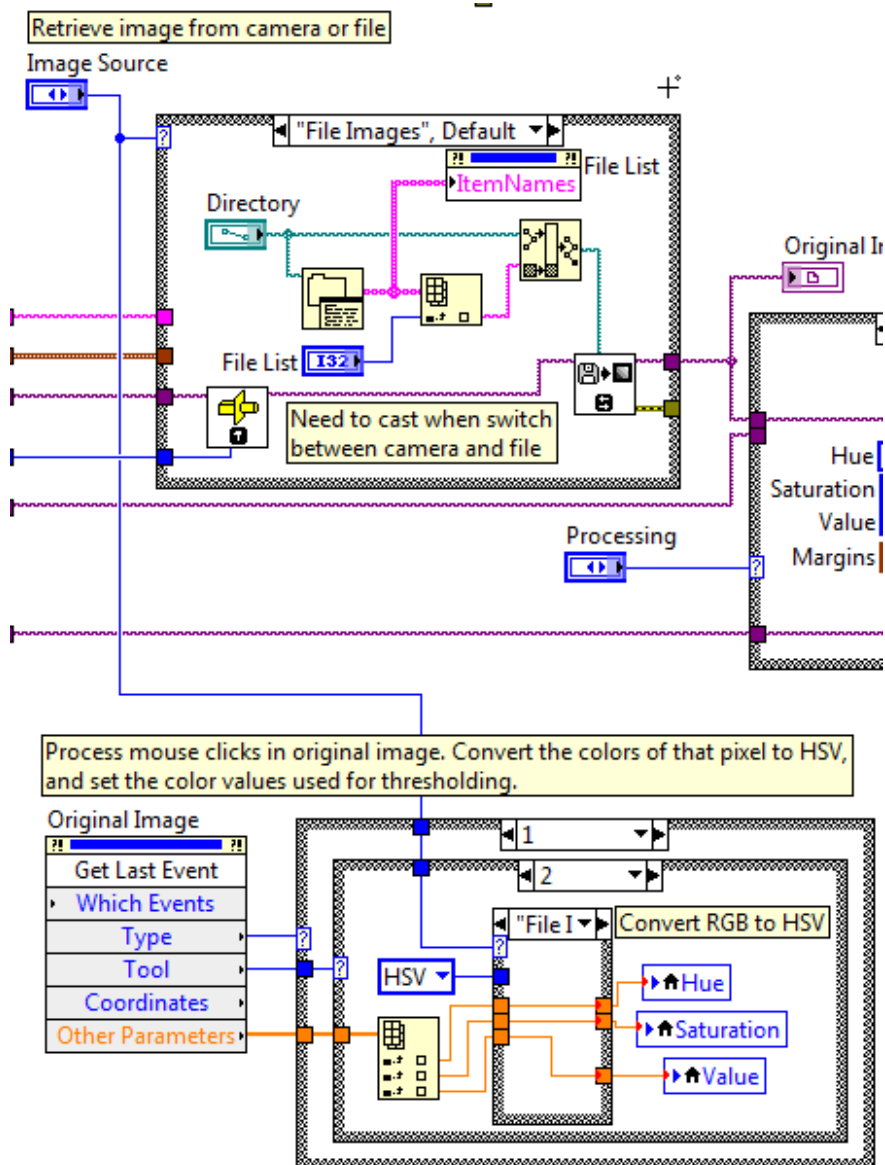
The left hand side of the LabVIEW code handles image acquisition and setting thresholds on a click event. It will typically not be necessary to modify this code.

Thresholding
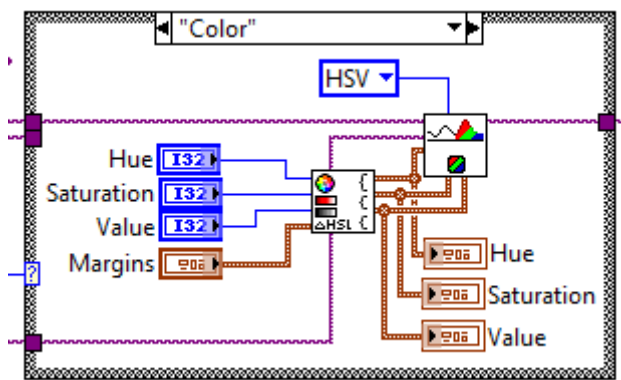
The thresholding operation occurs in the next Switch block, based on the selected mode, Color or Monochrome.

Scoring
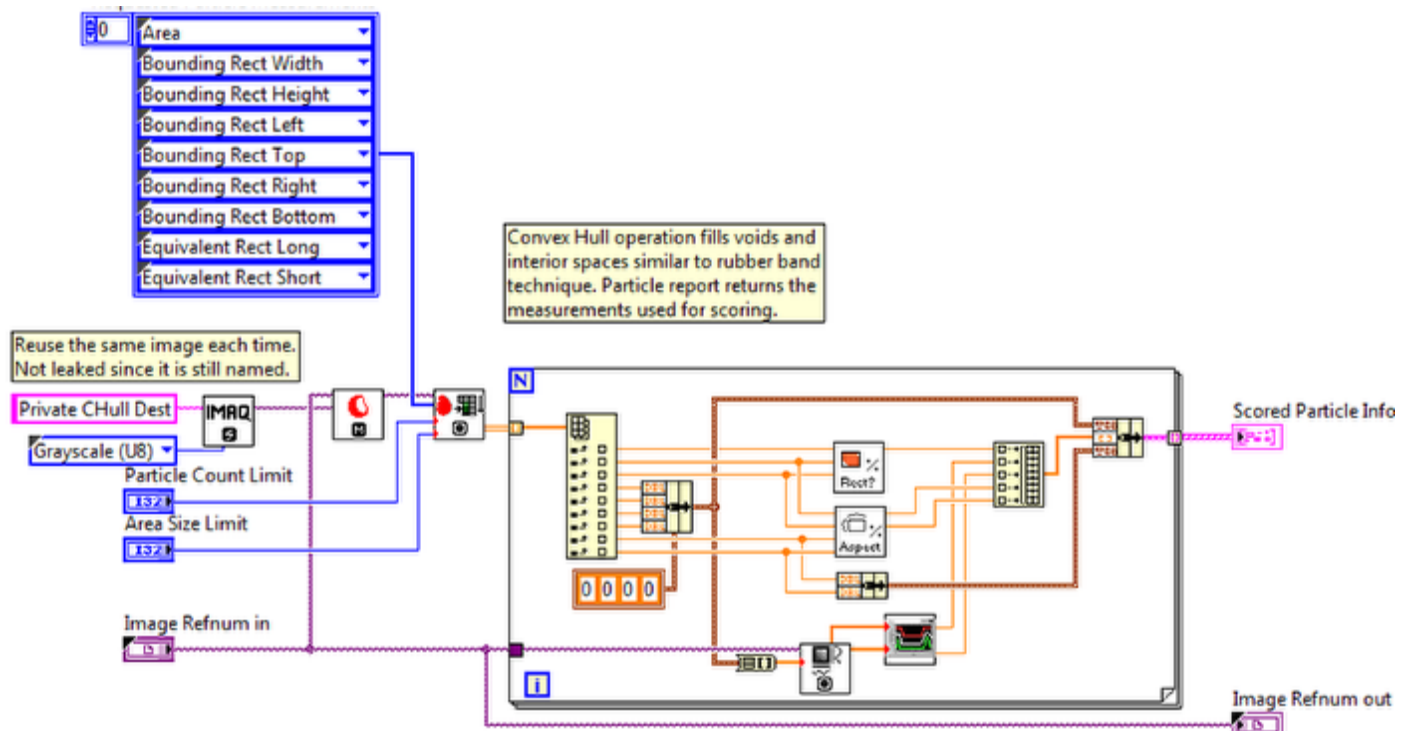
Particles are scored inside the Score Particles VI. First the Convex Hull operation is performed to fill in the hollow rectangles. Filtering for particle size and total number is integrated into the particle report generation contained in this VI. A report is generated for particles up to the maximum specified number if they meet the minimum specified area. The particles are then scored based on the rectangularity, aspect ration, and x and y edges. Note that the X and Y edge scores use the thresholded image and not the result of the convex hull. The bounding rect, and equivalent rect dimensions are passed out along with the scores for each particle for use in further operations.

Score Compare
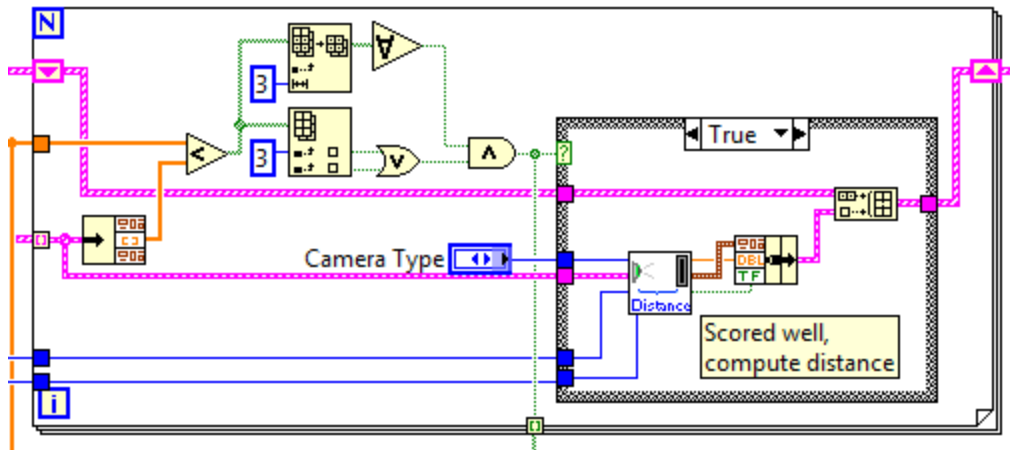
The scores for each particle are then compared against the specified score limits. The particle must exceed the minimum for the first three targets, and one of the two aspect ratio scores for the particle to be considered a target. If the particle is considered a target, then the distance is computed.

Compute Distance
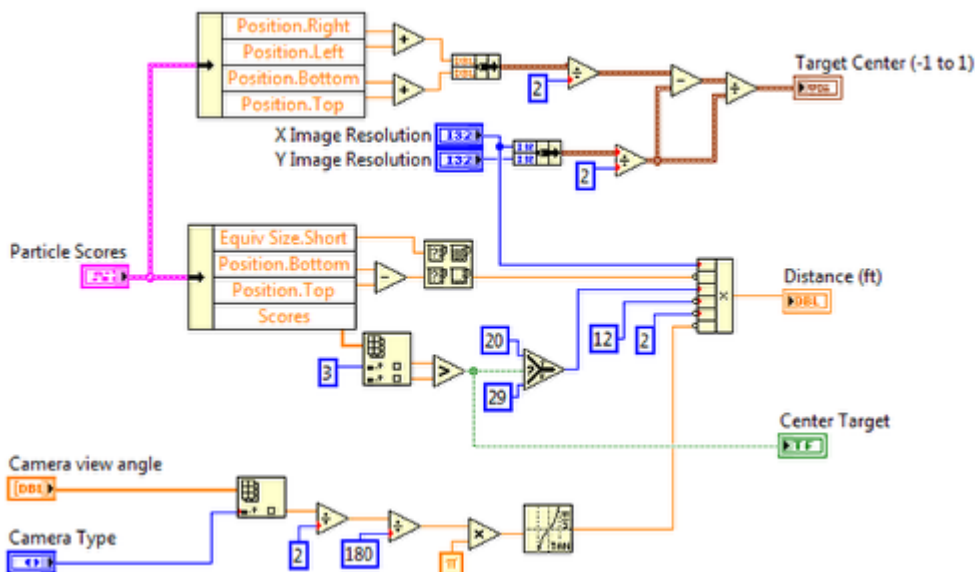
The code in the Compute Distance VI implements the formula described in the previous article. The top part of the VI normalizes the target bounding box to provide a coordinate pair for the target center on the (-1,1) scale. The bottom part of the VI computes the distance. Note that whichever aspect ratio score is the highest determines whether the target is considered to be a Center Target (High Goal) or not (Middle Goal). Also note that when using the block multiply VI, the inputs with "bubbles" indicate that the input is inverted before multiplying, the equivalent of a divide.

Sample Images

A number of sample images are provided in the example Project Directory. The provided images are broken up into two groups, one group was taken using the single green LED ring light available in FIRST Choice. The other group of images was taken using a variety of other LED ring light configurations, including a number of images taken with a smaller ring light nested inside a larger one.

C++/Java Code

The [Identifying the Targets](#) section explains a theoretical approach to locating the Vision Targets on the 2013 FRC Field. This document will cover the details of C++ and Java examples which implement this theoretical approach. Note that in addition to the typical differences between the C++ and Java WPILib code, there are also a few additional differences prompted by the way the NIVision functions are accessed from the Java code. Through the syntax may differ slightly the general approaches are similar enough that this document will walk through the C++ code which should provide sufficient insight into the function for both C++ and Java teams.

Finding the Example: C++
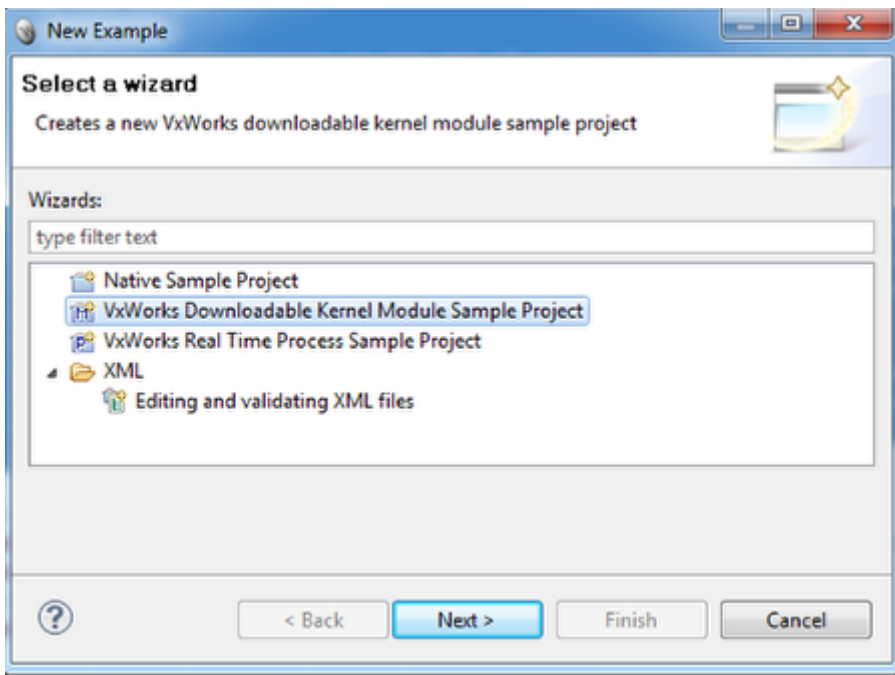
For C++ teams, the example can be found by selecting **File >> New >> Example.** Then select **VxWorks Downloadable Kernel Module Sample Project** and click **Next**. Select **FRC 2013 Vision Sample Program** and click **Finish** to open the sample.

Finding the Example: Java
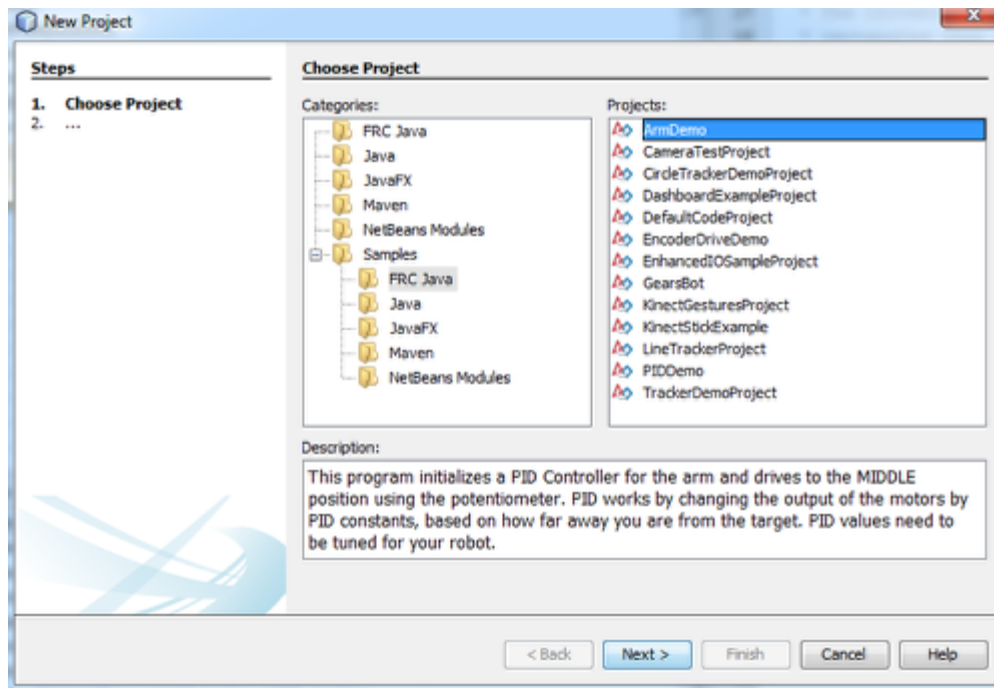
For Java teams, the example can be found by selecting **File >> New Project.** Then Expand the Samples folder, select FRC Java and click on the **2013VisionSampleProject**, then click **Next.** Enter a name and location for the project, then click **Finish**.

The Approach

Before examining the code, it is worth noting that the program samples are written using the Simple Robot framework and do not contain any other code in autonomous. The code will execute continuously, as quickly as possible, resulting in 100% usage of the cRIO CPU. When integrating this code into a different robot framework or integrating other team code, teams should make sure to add waits as appropriate and may wish to reduce the rate the code attempts to process at (processing every X loops in the Iterative Robot framework for example or every X milliseconds in the Command framework).

Also note that all of the methods are contained within the single class and file in order to make the example easier to read and understand. When adding to the example or integrating it with team code teams may wish to break out the scoring methods and structure into a separate class and file(s) in order to better organize the code.

Code Constants

```
//Camera constants used for distance calculation
#define X_IMAGE_RES 320      //X Image resolution in pixels, should be 160, 320 or 640
#define VIEW_ANGLE 48        //Axis 206 camera
//#define VIEW_ANGLE 43.5  //Axis M1011 camera
#define PI 3.141592653

//Score limits used for target identification
#define RECTANGULARITY_LIMIT 60
#define ASPECT_RATIO_LIMIT 75
#define X_EDGE_LIMIT 40
#define Y_EDGE_LIMIT 60

//Minimum area of particles to be considered
#define AREA_MINIMUM 500

//Edge profile constants used for hollowness score calculation
#define XMAXSIZE 24
#define XMINSIZE 24
#define YMAXSIZE 24
#define YMINSIZE 48
const double xMax[XMAXSIZE] = {1, 1, 1, 1, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, 1, 1, 1, 1};
const double xMin[XMINSIZE] = {.4, .6, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, .1, 0.6, 0};
const double yMax[YMAXSIZE] = {1, 1, 1, 1, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, .5, 1, 1, 1, 1};
const double yMin[YMINSIZE] = {.4, .6, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05,
                               .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05, .05,
                               .05, .05, .6, 0};
```

The sample code uses a number of constants that can be modified to tweak the behavior of the code. Many of these constants are defined at the top of the code, but teams should note that there are additional values contained inline that may also be tweaked such as the threshold values for the color threshold. Note that when changing camera resolutions, in addition to changing the resolution constant, it may also be necessary to change the Area Minimum constant to an appropriate value.

Scores Structure

```
struct Scores {
    double rectangularity;
    double aspectRatioInner;
    double aspectRatioOuter;
    double xEdge;
    double yEdge;

};
```

In order to store the scores for all of the individual tests for a particular particle together, a structure is used to contain all of the scores.

Filter Criteria

```
Threshold threshold(60, 100, 90, 255, 20, 255); //HSV threshold criteria, ranges are in that order ie. Hue is 60-100
ParticleFilterCriteria2 criteria[] = {
      {IMAQ_MT_AREA, AREA_MINIMUM, 65535, false, false}
};                                              //Particle filter criteria, used to filter out small particles
```

The threshold values used for the color threshold and the criteria object used for filtering out small particles is defined here. The filter criteria runs from the specified minimum area to the max integer value in order to filter out all particles smaller than the minimum.

Image Operations

```
ColorImage *image;
image = new RGBImage("/testImage.jpg");      // get the sample image from the cRIO flash
//camera.GetImage(image);                  //To get the images from the camera comment the line above and uncomment this one
BinaryImage *thresholdImage = image->ThresholdHSV(threshold);   // get just the green target pixels
//thresholdImage->Write("/threshold.bmp");
BinaryImage *convexHullImage = thresholdImage->ConvexHull(false);  // fill in partial and full rectangles
//convexHullImage->Write("/ConvexHull.bmp");
BinaryImage *filteredImage = convexHullImage->ParticleFilter(criteria, 1);  //Remove small particles
//filteredImage->Write("Filtered.bmp");
```

The first step of the processing is to perform the image operations: thresholding, convex hull and filtering. Code has been provided, but commented out, to write out each step of the image processing to the cRIO flash where it can be retrieved using FTP. To access and view the images, open up a Windows Explorer window and enter "FTP://10.XX.YY.2" in the navigation bar, where XXYY is a 4 digit FRC team number.

The code provided uses an image from cRIO flash named testImage.jpg, commented code is also provided to read from the Axis camera. For a set of sample images to start with see the "Sample Images" section at the end of this document. It is highly recommended to take a new set of sample images using the actual robot, camera and lighting when available.

Note: It is strongly recommended to comment out the while loop before enabling the image writes in order to preserve the cRIO flash memory. Writing the images within the while loop may result in excessive wear to the cRIO flash memory.

Particle Analysis

```
vector<ParticleAnalysisReport> *reports = filteredImage->GetOrderedParticleAnalysisReports();
scores = new Scores[reports->size()];
```

A report is generated for each particle and then the reports are ordered from largest particle to smallest. An array of Scores structures is created based on the number of particles found.

Scoring Particles

```
for (unsigned i = 0; i < reports->size(); i++) {
    ParticleAnalysisReport *report = &(reports->at(i));

    scores[i].rectangularity = scoreRectangularity(report);
    scores[i].aspectRatioOuter = scoreAspectRatio(filteredImage, report, true);
    scores[i].aspectRatioInner = scoreAspectRatio(filteredImage, report, false);
    scores[i].xEdge = scoreXEdge(thresholdImage, report);
    scores[i].yEdge = scoreYEdge(thresholdImage, report);

    if(scoreCompare(scores[i], false))
    {
        printf("particle: %d is a High Goal  centerX: %f  centerY: %f \n", i, report->center_mass_x_normalized, report->center_mass_y_normalized);
        printf("Distance: %f \n", computeDistance(thresholdImage, report, false));
    } else if (scoreCompare(scores[i], true)) {
        printf("particle: %d  is a Middle Goal  centerX: %f  centerY: %f \n", i, report->center_mass_x_normalized, report->center_mass_y_normalized);
        printf("Distance: %f \n", computeDistance(thresholdImage, report, true));
    } else {
        printf("particle: %d  is not a goal  centerX: %f  centerY: %f \n", i, report->center_mass_x_normalized, report->center_mass_y_normalized);
    }
    printf("rect: %f  ARinner: %f \n", scores[i].rectangularity, scores[i].aspectRatioInner);
    printf("ARouter: %f  xEdge: %f  yEdge: %f  \n", scores[i].aspectRatioOuter, scores[i].xEdge, scores[i].yEdge);
}
```

Each particle is scored according to the approach described in the Identifying the Targets section, then the scores are compared to the defined minimum scores for both a High Goal target and Middle Goal target. If a particle is considered to be a target, the distance to the target (in inches) is computed and information about the target is printed out to the console. If the particle is not a target a more limited set of debug information is printed to the console.

This section of code is one that teams are recommended to modify to suit their robot and approach to vision processing and debugging, Teams may wish to modify the information printed to the console, or replace the console code with SmartDashboard code for the same purpose. Once the targets have been identified teams can use the distance, the normalized center, or both in order to react to the target position.

Cleanup

```
// be sure to delete images after using them
delete filteredImage;
delete convexHullImage;
delete thresholdImage;
delete image;

//delete allocated reports and Scores objects also
delete scores;
delete reports;
```

After processing is complete it is critical to release the memory from dynamically allocated objects such as the images, array of scores and vector of reports. Failing to release the memory used by these objects will "leak" the references to this memory and will result in the memory usage of the program steadily climbing until no free memory remains and the program crashes.

Sample Images

A number of sample images have been provided which were taken using an Axis 206 camera set up as recommended in the Camera Settings section. These images can be found in the VisionImages folder inside the project directory for the sample project. The images are split into two directories, one directory was taken with the green LED ring light available from FIRST Choice. The other images were taken with a variety of other LED light configurations, many of which had a smaller light nested inside a larger one.