

VISION PROCESSING

Table of Contents

2014 Vision Processing.....3

 Target Info and Retroreflection.....4

 Camera Settings..... 10

 Identifying and Processing the Targets..... 16

 Calibration 24

 LabVIEW Code 33

 C++/Java Code 44

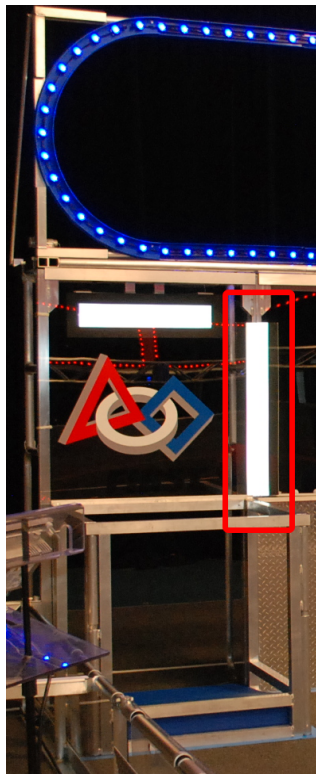
 Axis M1013 Camera Compatibility..... 53

2014 Vision Processing

Target Info and Retroreflection

This document describes the Vision Targets from the 2014 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](#)

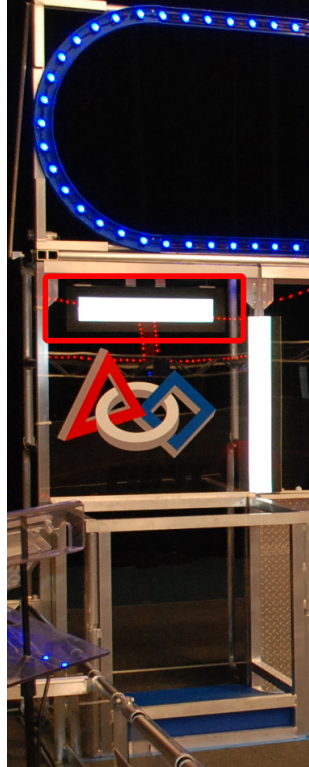
Vertical Targets



The Vertical Vision Target consists of 4" wide, 32" tall stripe of retroreflective material (3M 8830 Silver Marking Film) bordered by 2in. wide black gaffers tape on the left and right sides. The Vertical Targets are located behind the polycarbonate (above the low goal) and acrylic (in front of the Player Station) sheets above the inside edge of each Low Goal, approximately 37.5" above the carpet. When properly lit, the retroreflective tape produces a bright and/or color-saturated marker.

Vision Processing

Horizontal Targets



The Horizontal Target consists of a 23.5" long, 4" tall stripe of retroreflective material bordered on all sides by a 2" black ABS plastic frame. The Horizontal Targets are centered above the Low Goal, behind the polycarbonate sheet. The reflective material begins 68" above the carpet.

Horizontal Target Behavior

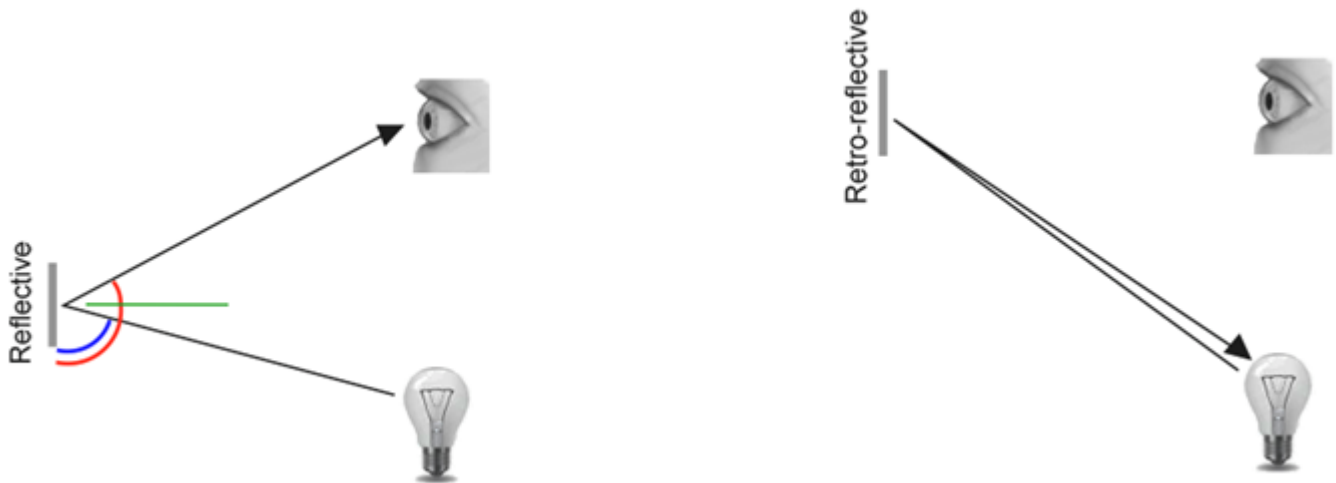
The Horizontal Target is a dynamic field element which is actuated during autonomous to indicate whether the corresponding High and Low goals are Hot or not. Before the match starts, both targets will have the reflective material showing out towards the field.

- When the match starts, one target will actuate to point the reflective material upwards, hiding it from view. The target which flips first will be randomly selected, but will always be the same for both alliances from the robot perspective. This means that the targets active at the same time will be located diagonally across the field from each other. The goal which has the reflective material showing is the Hot goal.
- Halfway through the autonomous period both targets will flip their state indicating that the other goal is now Hot.
- At the end of the autonomous period both targets will return to showing the material. The targets will remain in this state (both showing) for the duration of the Teleoperated period.

Vision Processing

The targets are actuated by an electric solenoid and have state transition times of ~?.? seconds.

Retroreflectivity vs. Reflectivity

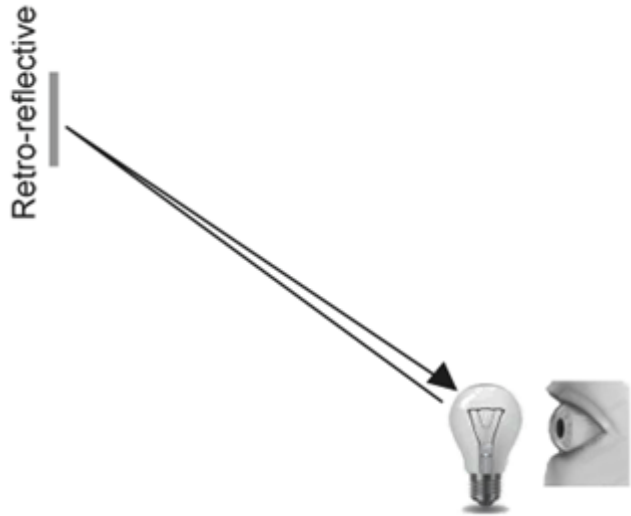


Highly reflective materials are generally mirrored so that light “bounces off” at a supplementary angle. As shown above-left, the blue and red angles sum to 180 degrees. An equivalent explanation is that the light reflects about the surface normal the green line drawn perpendicular to the surface. Notice 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.

Vision Processing

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

Vision Processing

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 above. 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](#)

Vision Processing

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 2014 field, see the documents linked from [this page](#).

Sample Images

Sample Images are packaged with the examples for each language. The locations are described in the [LabVIEW Code](#) and [C++/Java Code](#) articles. The images included with the examples do not have the yellow LEDs lit on the goal above the target. Some images with the LEDs lit have been uploaded here: http://firstforge.wpi.edu/sf/frs/do/viewRelease/projects.wpilib/frs.2014_vision_images.2014_vision_images_supplement

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.

Changing Camera Settings

The screenshot displays the Axis camera settings web interface. The left sidebar contains navigation links: Basic Setup (M1013), Video (with sub-links Video Stream, Stream Profiles, Camera Settings, Overlay Image, and Privacy Mask), Live View Config, Detectors, Events, Recordings, System Options, and About. The main content area is divided into two sections. The top section, 'Camera Settings', includes 'View Area' (with an 'Enable View Area' checkbox), 'Image Appearance' (sliders for Color level, Brightness, Sharpness, and Contrast, all set to 50), 'White Balance' (set to 'Fixed Outdoor 1'), 'Exposure Settings' (Exposure value slider at 50, 'Enable Backlight compensation' checkbox), and 'Exposure priority' (set to 'Default'). The bottom section, 'Image Settings', includes 'Image Appearance' (Resolution: 640x480, Compression: 30, Rotate image: 0 degrees, Color level: 50, Brightness: 50, Sharpness: 0), 'Overlay Settings' (checkboxes for 'Include date', 'Include time', and 'Include text'), and 'Video Stream' (text input for stream name).

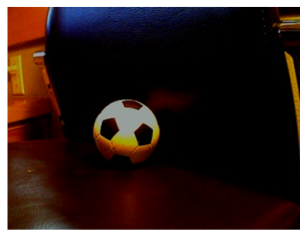
To change the camera settings on any of the supported Axis cameras (206, M1011, M1013), browse to the camera's webpage by entering its address (usually 10.TE.AM.11) in a web browser. Click **Setup** near the top right corner of the page. On the M1013, the settings listed below are split between the **Video Stream** page and the **Camera Settings** page, both listed under the **Video** section.

Vision Processing

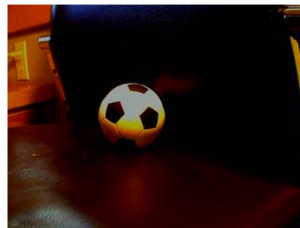
Focus

The Axis M1011 has a fixed-focus lens and no adjustment is needed. The Axis 206 camera has a black bezel around the lens that rotates to move the lens in and out and adjust focus. The Axis M103 has a silver and black bezel assembly around the lens to adjust the focus. Ensure that the images you are processing are relatively sharp and focused for the distances needed on your robot.

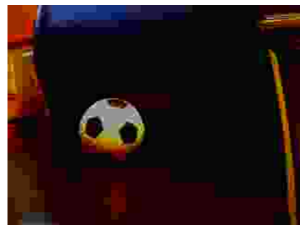
Compression



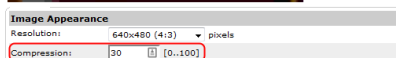
320x240 Color Image:
Compression set to 0. Image file size is 20,715 bytes.
High quality, but large in size. May be slower to compress and decompress - which impacts frame rate.



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 from 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.

Vision Processing

Resolution

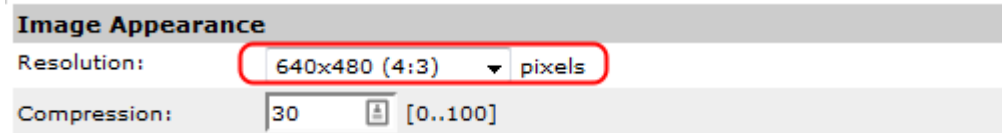


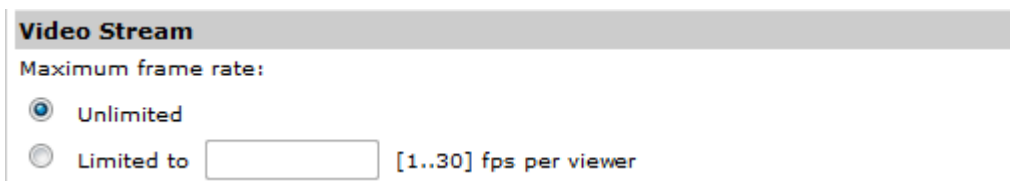
Image sizes shared by the supported cameras are 160x120, 320x240, and 640x480. The M1011 and 1013 have additional sizes, but they aren't built into WPILib. The largest image size has four times as many pixels that are one-fourth the size of the middle size image. The large image has sixteen times as 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.

Note: When requesting images using LabVIEW (either the Dashboard or Robot Code), the resolution and Frame Rate settings of the camera will be ignored. The LabVIEW code specifies the framerate and resolution as part of the stream request (this does not change the settings stored in the camera, it overrides that setting for the specific stream). The SmartDashboard and robot code in C++ or Java will use the resolution and framerate stored in the camera.

Frame Rate



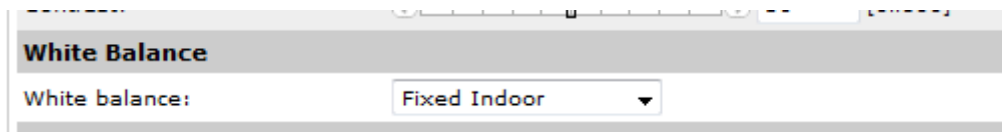
The Axis Cameras have a max framerate of 30 frames per second. If desired, a limit can be set lower to reduce bandwidth consumption.

Vision Processing

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.

Vision Processing

Exposure

Image Appearance M1013

Color level: [Slider] 50 [0..100]

Brightness: [Slider] 0 [0..100]

Sharpness: [Slider] 50 [0..100]

Contrast: [Slider] 50 [0..100]

White Balance

White balance: Fixed Indoor

Exposure Settings

Exposure value: [Slider] 0 [0..100]

Enable Backlight compensation: ☐

Exposure priority: Motion

Lighting Conditions 206/M1011

White balance: Automatic

Exposure control: Automatic

Low Light Behavior

Exposure priority: None

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 on a 206 or M1011, 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.

Vision Processing

The M1013 exposure settings look a little different. The Enable Backlight compensation option is similar to the Auto exposure settings of the M1011 and 206 and you will usually want to un-check this box. Adjust the Brightness and Exposure value sliders until your image looks as desired. The Exposure Priority should generally be set to Motion. This will prioritize framerate over image quality. Note that even with these settings the M1013 camera still performs some auto exposure compensation so it is recommended to check calibration frequently to minimize any impact lighting changes may have on image processing. See the article on Calibration for more details.

Identifying and Processing the Targets

Once an image is captured, the next step is to identify Vision Target(s) in the image. This document will walk through one approach to identifying the 2014 targets and distinguishing between targets for Hot and not Hot 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

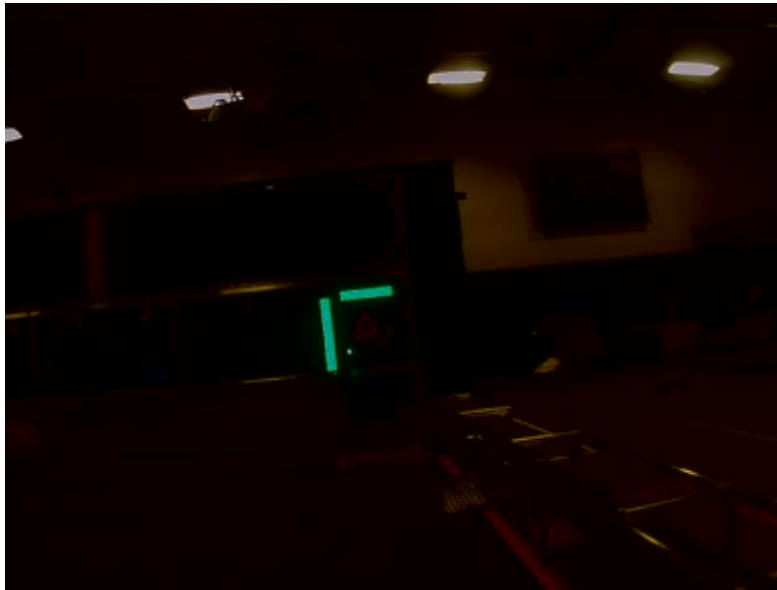
This document walks through the approach used by the example code provided in LabVIEW (for PC or cRIO), C++ and Java, details on the language specific examples are provided in subsequent articles. In addition to these options teams should be aware of the following alternatives that allow for vision processing 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.

Vision Processing



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.

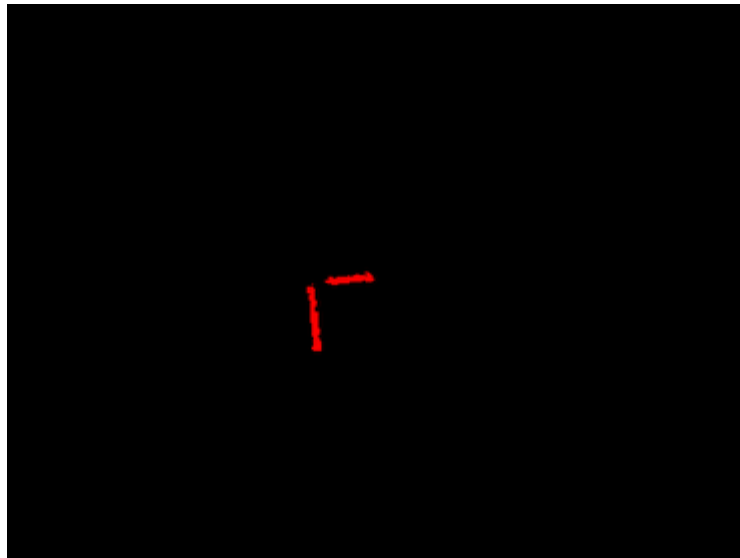
The example code uses the HSV color space to specify the color of the target. The primary reason is that it readily allows for using the brightness of the targets relative to the rest of the image as a filtering criteria by using the Value (HSV) or Luminance (HSL) component. Another reason to use the 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.

Vision Processing

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.



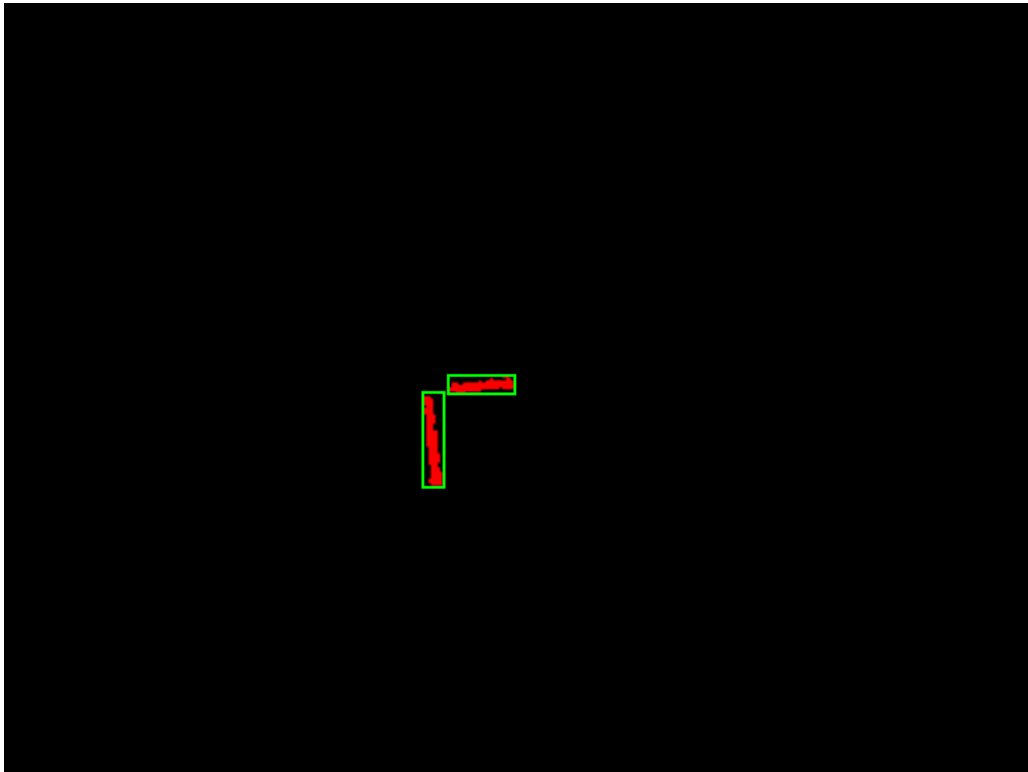
Particle Analysis

After the masking 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 $(\text{Particle Area} / \text{Bounding Box Area}) * 100$. A perfectly rectangular particle will score 100, a circular particle will score $(\pi/4) * 100$, or about 78, and the score will drop even further for diagonal lines and other streaks. In this image the camera is slightly tilted causing the rectangles to be slightly skewed. This results in a diminished rectangularity score, but the resulting scores are still high enough to be considered a target. In the images below a representative bounding box for each particle has been drawn in green over the masked image.

Vision Processing



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 Horizontal and Vertical targets on the field have different aspect ratios, so two aspect ratio scores are generated, one for each target type. The field target ratios are $4/32$ for the vertical and $23.5/4$ for the horizontal target. 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.

Target Pairing

After identifying which particles appear to be targets, the next step is to determine if there are any complete Hot Goal targets in the image by attempting to pair each detected vertical target with each detected horizontal target and calculating some scores to determine if they are likely part of the same Hot Goal indicator.

Vision Processing

Left/Right Score

The first calculation is to check if the horizontal target appears to be in the correct horizontal location relative to the vertical target. To determine this, the distance between the center of the horizontal target and the closest edge of the vertical target is calculated. This distance should be approximately 1.2 times larger than the width of the horizontal target. This ratio is then converted to a 0-100 score using a piecewise linear function that goes from (0,0) to (1,100) to (2,0) and is 0 for all values outside the range 0 to 2.

Tape Width Score

If the two targets are located physically close to each other the width of the tape should appear very similar to the camera. The ratio of the vertical target height to the horizontal target width is calculated and converted to a 0-100 score using the same method described above.

Vertical Score

The last score checks if the horizontal target is in the proper vertical location relative to the vertical target. The difference between the top of the vertical target and the center of the horizontal target is calculated and divided by 4 times the height of the horizontal target. 1 minus this value is used to calculate the score as indicated above.

Hot Targets

The implementation in the LabVIEW code differs slightly from the C++/Java example at this point. The LabVIEW code determines the best horizontal match for each vertical target, checks if it is a hot target, then provides targeting information for all vertical targets, sorted in the order Left Hot targets, Right Hot targets, Not Hot targets. The C++ and Java example selects the one target pair with the best total score, then checks if it is a Hot target or not.

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

Vision Processing

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.

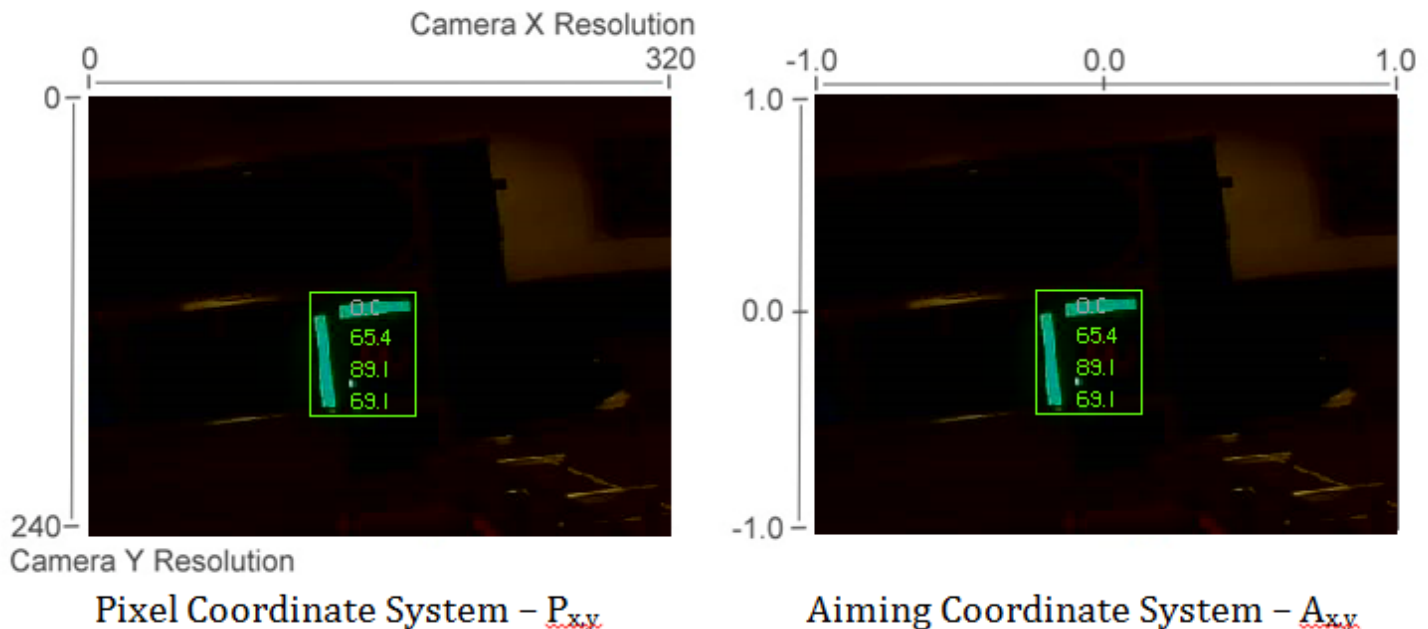
To convert a point from the pixel system to the aiming system, we can use the formula shown below.

The resulting coordinates are close to what you may want, but the Y axis is inverted. This could be corrected by multiplying the point by [1,-1] (Note: this is not done in the sample code). This coordinate system is useful because it has a centered origin and the scale is similar to joystick outputs and RobotDrive inputs.

Note: In the C++ and Java example, this information is provided by using the Normalized Center of mass from the target report for the horizontal or vertical target particle.

Note 2: In LabVIEW this information is calculated using the vertical target in order to apply to both Hot and Not Hot targets.

$$A_{x,y} = (P_{x,y} - \frac{resolution_{x,y}}{2}) / \frac{resolution_{x,y}}{2}$$



Distance

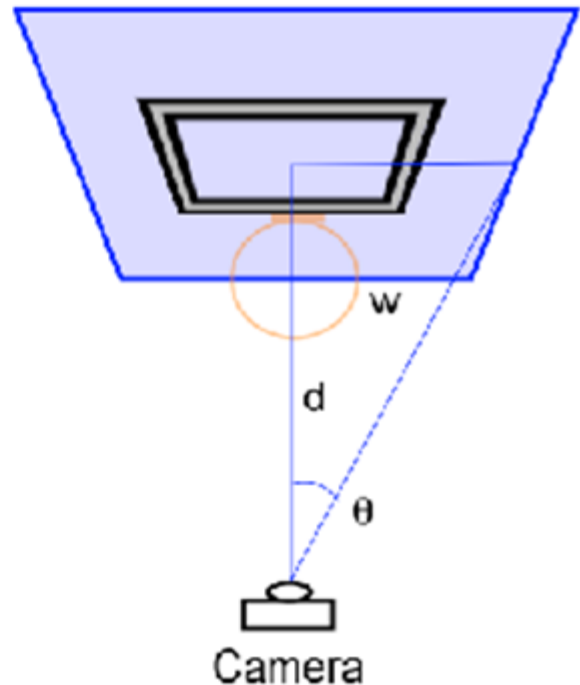
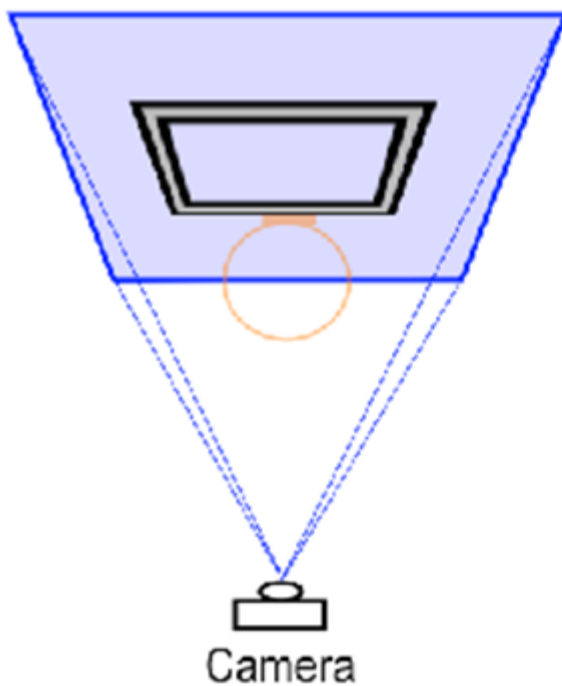
The target distance is computed with knowledge about the target size and the camera optics. The approach uses information about the camera lens view angle and the width of the camera field of

Vision Processing

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 ...

$$\tan\theta = w/d$$

The datasheets for the Axis cameras can be found at the following URLs: [Axis 206](#), [AxisM1011](#), [Axis M1013](#). These give rough horizontal view angles for the lenses. Remember that this is for entire field of view, and is therefore 2θ . This year's code uses the vertical field-of-view and it is therefore highly recommend to perform calibration (as described in the next article) to determine the appropriate view angle for your camera (empirically determined values for each camera type are included in the code as a reference).



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 relationships of ...

$$T_{ft}/T_{pixel} = FOV_{ft}/FOV_{pixel} \text{ and } FOV_{ft} = 2*w = 2*d*\tan\theta$$

Vision Processing

to create an equation to solve for d , the distance from the target:

$$d = T_{ft} * FOV_{pixel} / (2 * T_{pixel} * \tan \Theta)$$

The Y axis field of view is calculated. We know that the target height measures 32 in. for the Vertical Target, and in the example images used earlier the Vertical target rectangle measures 40 pixels when the camera resolution was 320x240. This means that the blue rectangle width is $2.66 * 320 / 30$ or 21.28ft. Half of the width is 10.64 ft, and the camera used was the 206, so the view angle is $\sim 41.7^\circ$, making Θ be 20.85° . Putting this information to use, the distance to the target is equal to $10.64 / \tan 20.85^\circ$ or 28ft.

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 41.7° for the 206, 37.4° for the M1011, and 49° for the M1013 gave better results. Information on performing your own distance calibration is included in the next article.

Calibration

While many of the numbers for the Vision Processing code can be determined theoretically, there are a few parameters that are typically best to measure empirically then enter back into the code (a process typically known as calibration). This article will show how to perform calibration for the Color (masking), and View Angle (distance) using the NI Vision Assistant. If you are using C++ or Java and have not yet installed the NI Vision Assistant, see the article Installing NI Vision Assistant.

Enable Snapshots

AXIS M1013 Network Camera

Live View | Setup | Help

Basic Setup

Video

Live View Config

Layout

Detectors

Events

Recordings

System Options

About

Live View Layout

Stream Profile

Stream profile: Motion JPEG

Show stream profile selection

Default Viewer

Windows Internet Explorer: AMC (ActiveX)

Other Browsers: Server push

Note: QuickTime is only used with H.264. Motion JPEG will be shown with AMC in Windows Internet Explorer and with server push in other browsers.

Viewer Settings

Show viewer toolbar

Enable H.264 decoder installation

Show crosshair in PTZ joystick mode

Use PTZ joystick mode as default

Enable recording button

* Not applicable to AMC (ActiveX).

Action Buttons

Show manual trigger button

Show snapshot button

User Defined Links

Show custom link 1

Name: Custom link 1

Use as: cgi link web link

URL: http://

Show custom link 2

Name: Custom link 2

Use as: cgi link web link

URL: http://

Show custom link 3

Name: Custom link 3

Use as: cgi link web link

URL: http://

Show custom link 4

Name: Custom link 4

Use as: cgi link web link


URL: http://

Save Reset

To capture snapshots from the Axis camera, you must first enable the Snapshot button. Open a web-browser and browse to camera's address (10.TE.AM.11), enter the Username/Password combo FRC/FRC if prompted, then click Setup->Live View Config->Layout. Click on the checkbox to Show snapshot button then click Save.

Vision Processing

Check Camera Settings

**AXIS 206 Network Camera**

[Live View](#) | [Setup](#) | [Help](#)

Basic Configuration


Video & Image

Video & Image

Advanced

Live View Config

System Options

Language 

About


Image Settings

Image Appearance

Resolution:

640x480

 pixels

Compression:

30

 [0..100]

Rotate image:

0

 degrees

Color level:

50

 [0..100] *

Brightness:

50

 [0..100] (Does not affect Test image)

Sharpness:

0

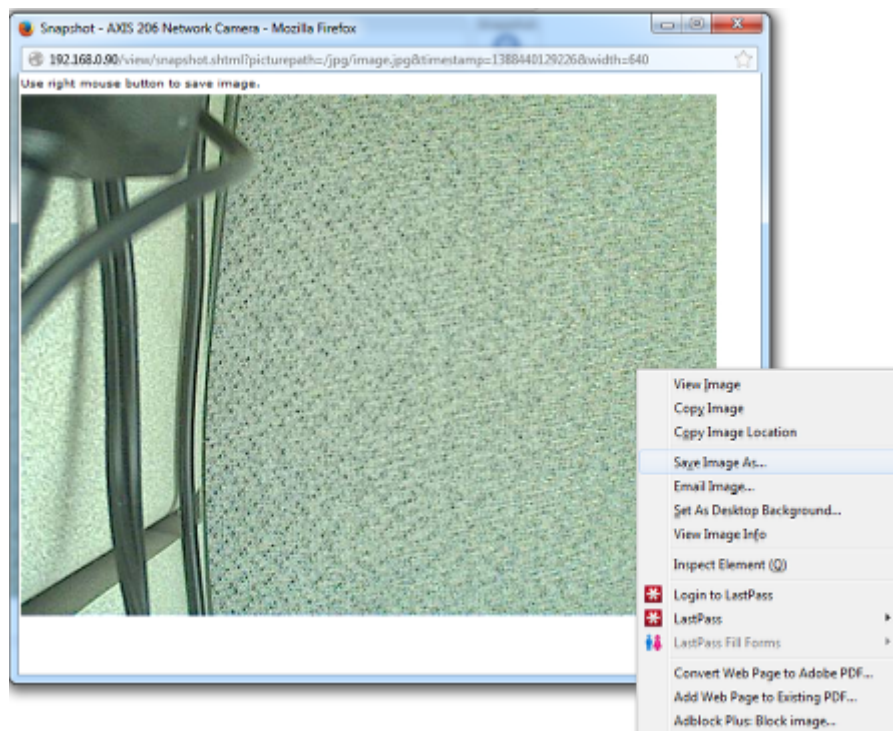
 (Does not affect Test image)

* Changes to color level do not affect Test image (exception 0 = B/W)

Depending on how you are capturing the image stream in your program, it may be possible to stream a different resolution, framerate and/or compression than what is saved in the camera and used in the Live View. Before performing any calibration it is recommended you verify that the settings in the camera match the settings in your code. To check the settings in the camera, click on the Video and Image header on the left side of the screen, then click Video and Image.

Vision Processing

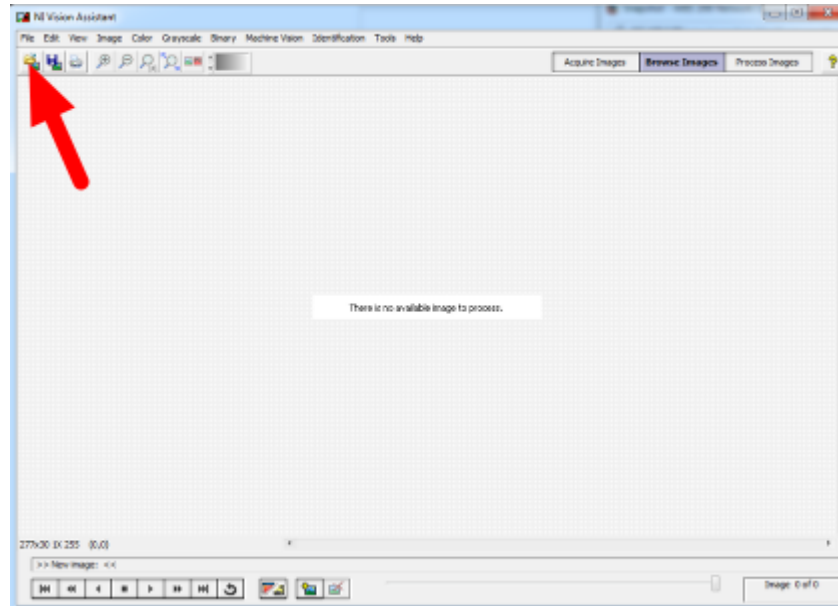
Capture Images



Click the Live View button to return to the Live View page and you should now see a Snapshot button. Clicking this button opens a pop-up window with a static image capture. Right-click on this image, select Save Image as and select your desired location and file name, then save the image.

Vision Processing

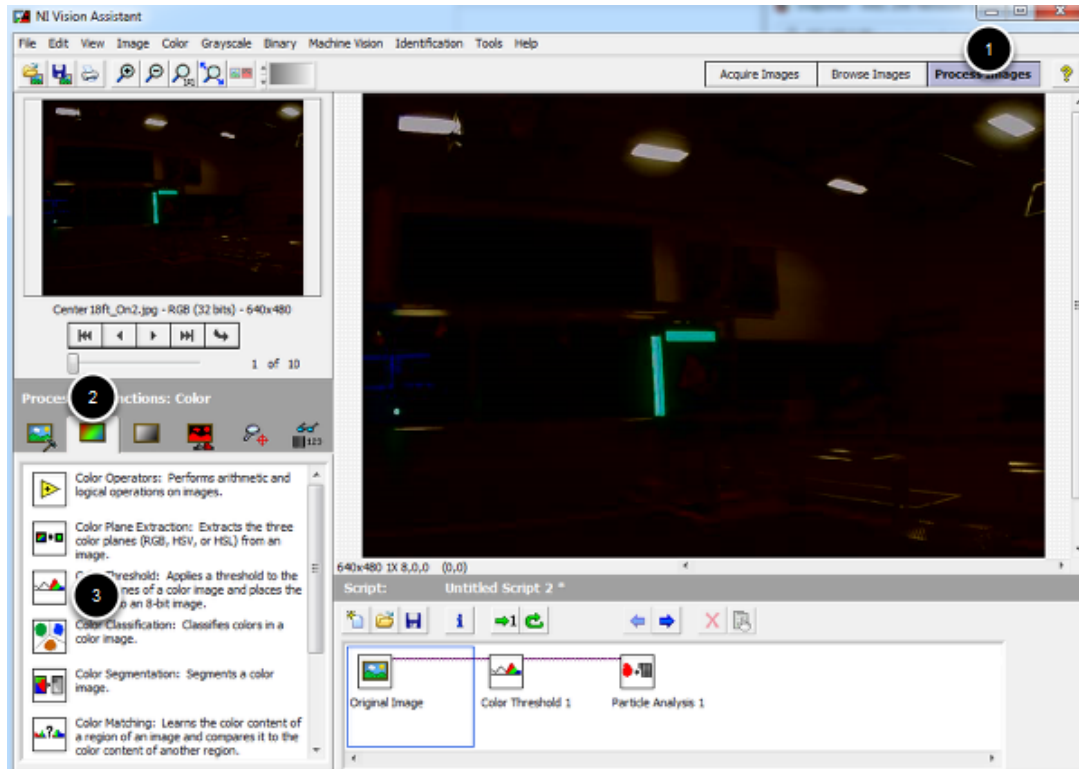
Load Image(s) in Vision Assistant



Open the NI Vision Assistant and select the Browse Images option. Select the Open Images icon in the top left of the Toolbar, then locate your images. Repeat as necessary to load all desired images.

Vision Processing

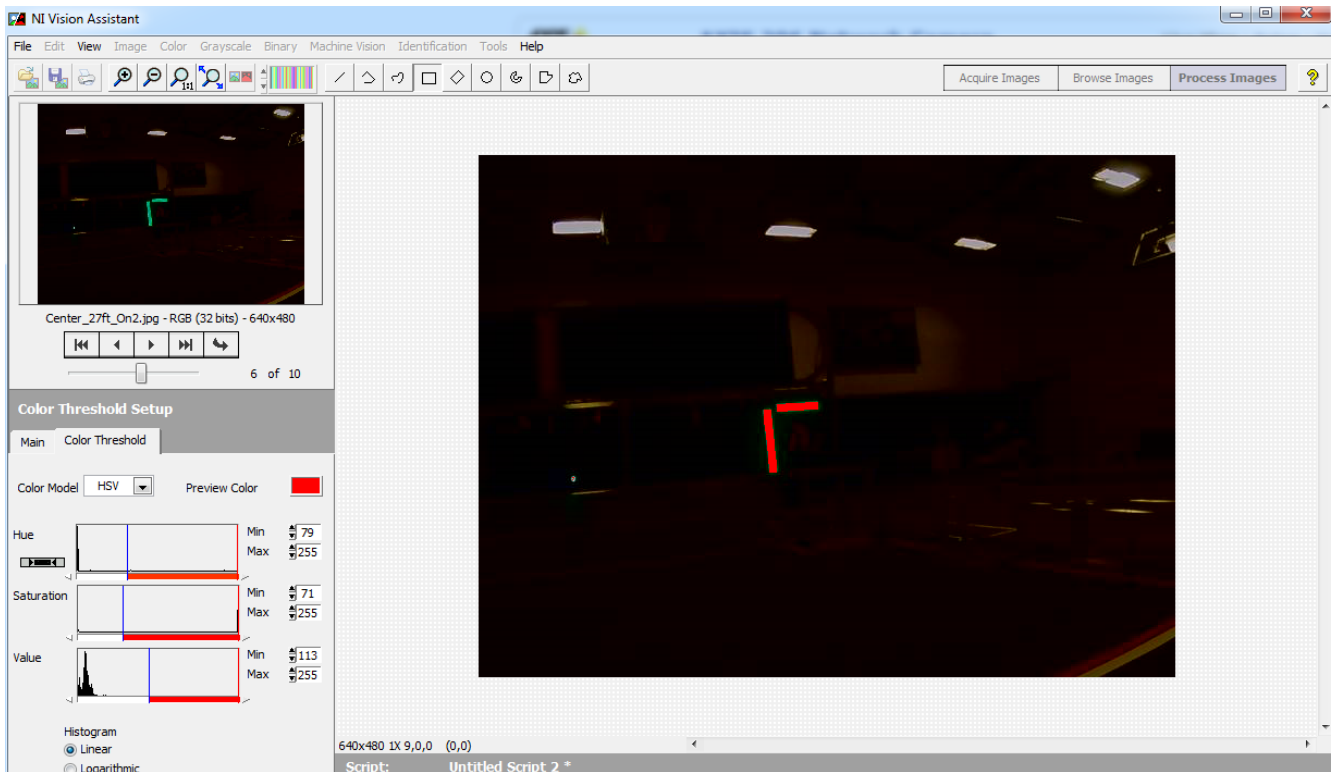
Color Threshold



Click Process Images in the top right, then select the color tab on the bottom right and click the Color Threshold icon.

Vision Processing

HSV Calibration



Change the Color Model dropdown to HSV. Next tune the window on each of the three values to cover as much of the target as possible while filtering everything else. If using a green light, you may want to use the values in the sample code as a starting point. If you have multiple images you can use the controls in the top left to cycle through them. Use the center two arrow controls or the slider to change the preview image in the top left window, then click the right-most arrow to make it the active image. When you are happy with the values you have selected, note down the ranges for the Hue, Saturation and Value. You will need to enter these into the appropriate place in the vision code. Click OK to finish adding the step to the script.

You may wish to take some new sample images using the time for camera calibration at your event to verify or tweak your ranges slightly based on the venue lighting conditions.

View Angle/Distance Calibration

While a theoretical view angle for each camera model can be found in the datasheet, empirical testing has found that these numbers may be a bit off even for the horizontal view angle. Given that this year's code uses the vertical field-of-view it is best to perform your own calibration for your camera (though empirical values for each camera type are included in the code as a reference). To do this set up an equation where the view angle, Θ , is the only unknown. To do this,

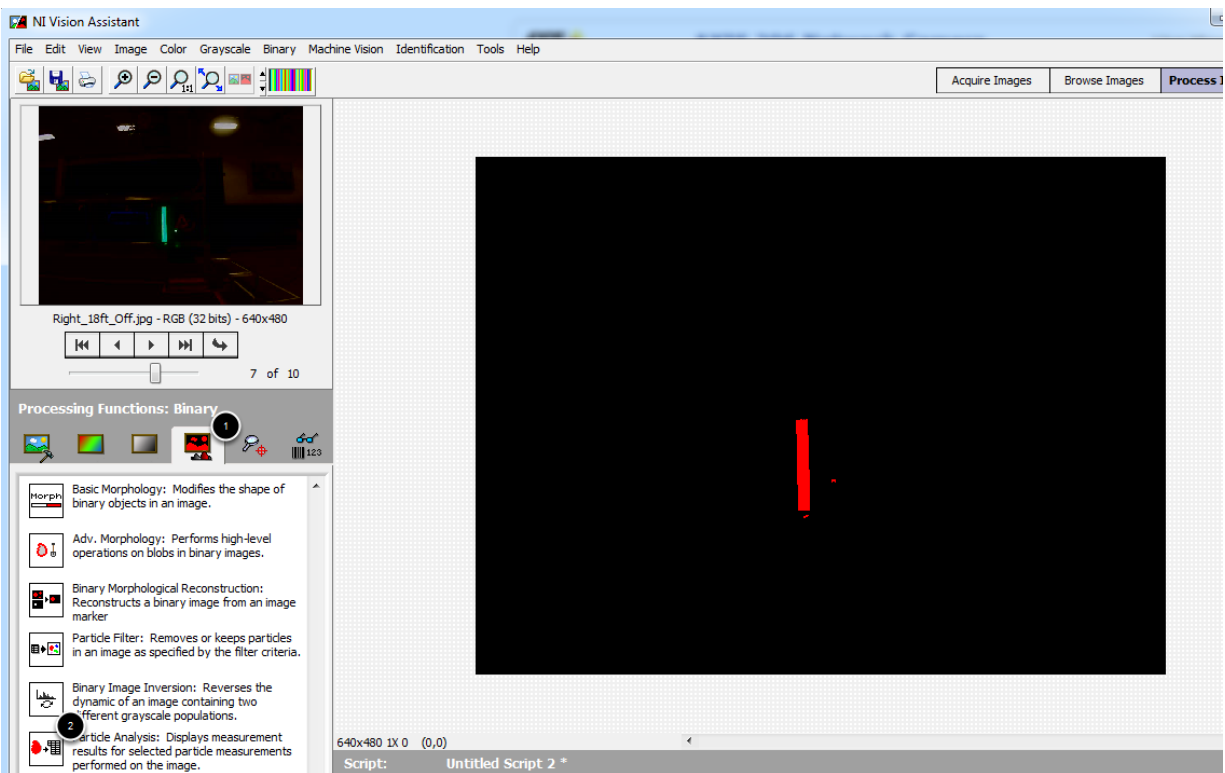
Vision Processing

utilize a target of known size at a known distance, leaving the view angle as the only unknown. Let's take our equation from the previous article, $d = T_{ft} * FOV_{pixel} / (T_{pixel} * \tan \Theta)$, and re-arrange it to solve for Θ :

$$\tan \Theta = T_{ft} * FOV_{pixel} / (T_{pixel} * d)$$

$$\Theta = \arctan(T_{ft} * FOV_{pixel} / (T_{pixel} * d))$$

Taking measurements

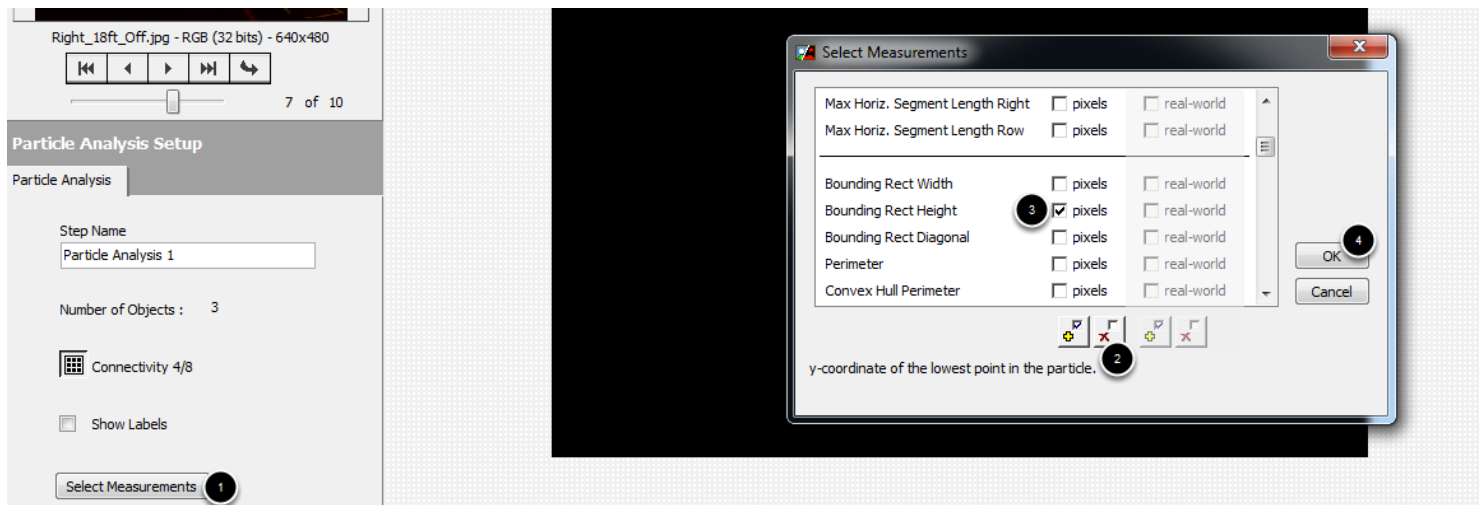


One way to take the required measurements is to use the same images of the retro-reflective tape that were used for the color calibration above. We can use Vision Assistant to provide the height of the detected blob in pixels. By measuring the real-world distance between the camera and the target, we now have all of the variables to solve our equation for the view angle.

To measure the particles in the image, click the Binary tab, then click the Particle Analysis icon.

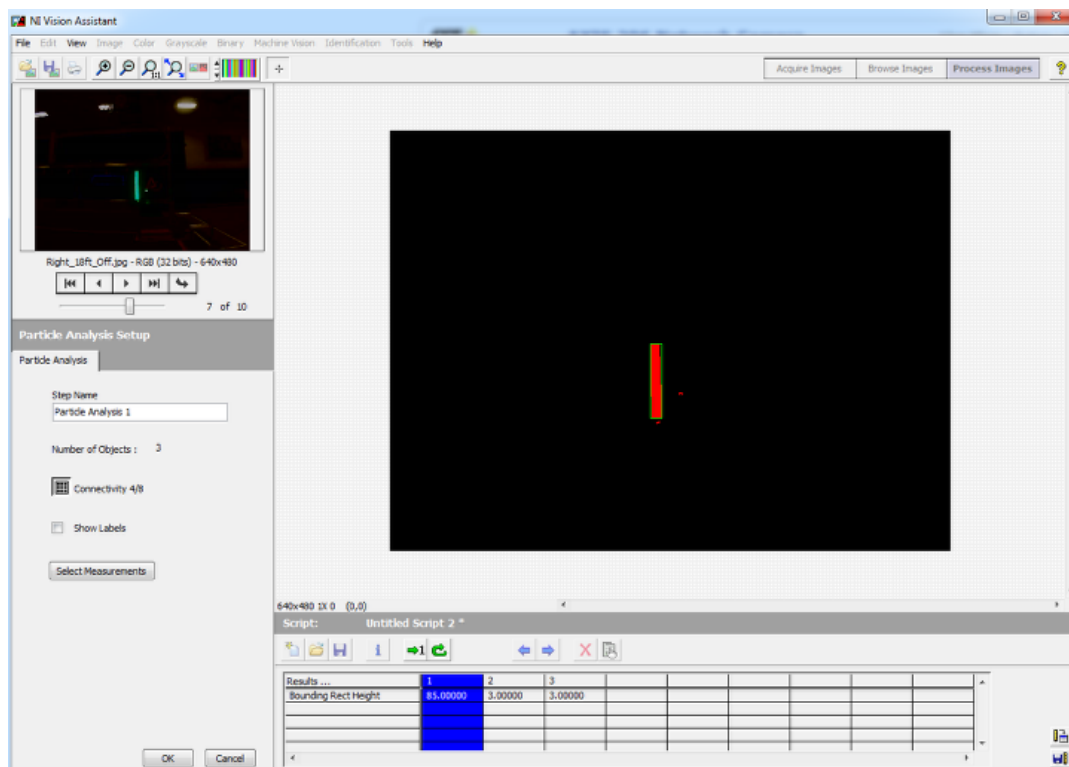
Vision Processing

Selecting Measurements



Click on the Select Measurements button. In this case, we are only interested in the bounding box height. Click on the button with the X to deselect all measurements, then locate the Bounding Rect Height measurement and check the box. Click OK to save.

Measuring the Particle



Vision Processing

The measurements for each particle will now be displayed in the window at the bottom of the screen. If your image has multiple particles, you can click in each box to have Vision Assistant highlight the particle so you can make sure you have the right one. This article will show the calculation using a single image, but you may wish to perform the calculation on multiple images from multiple distances and use a technique such as averaging or least squares fit to determine the appropriate value for the View angle. You can use the same arrow controls described in the color section above to change the active image.

Calculation

As seen in the previous step, the particle representing the 32in tall vertical target in this example measured 85 pixels tall in a 640x480 image. The image shown was taken from (very roughly) 18 ft. away. Plugging these numbers into the equation from above....

$$\Theta = \arctan(2.66*480/(2*85*18)) = 22.65 \text{ degrees}$$

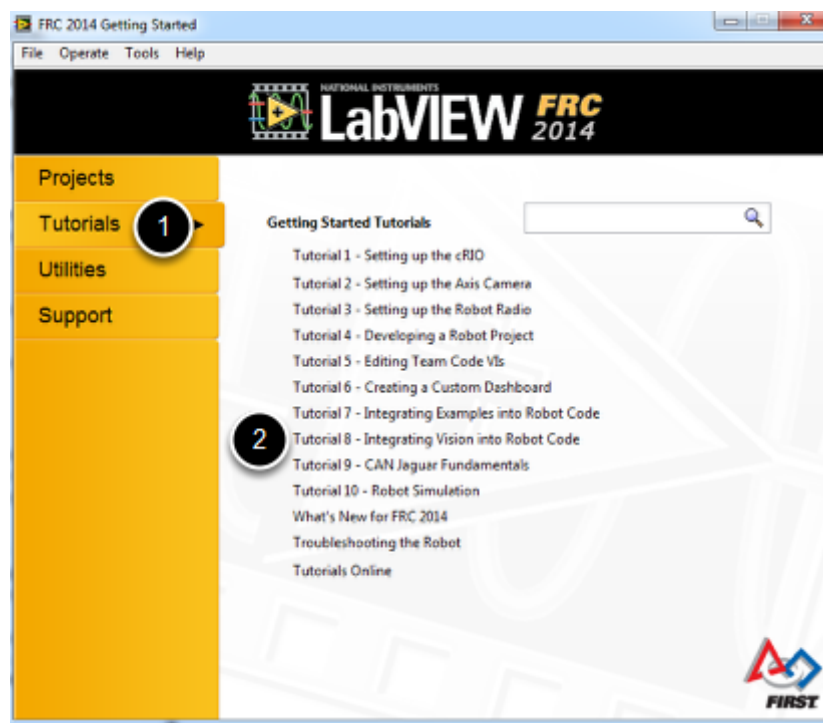
Depending on what you use to calculate the arctangent, your answer may be in radians, make sure to convert back to degrees if entering directly into the sample code as the view angle.

Note: The code uses **View Angle** and we just calculated Θ . Make sure to multiply Θ by 2 if replacing the constants in the code. Multiplying our result by 2 yields 45.3 degrees. This image is from a M1013 camera, so our value is a bit off from the previously measured 29.1 but given that the 18ft. was a very rough measurement this shows that we are in the ballpark and likely performed the calculation correctly.

LabVIEW Code

The previous articles detailed the theoretical approach to identifying the Vision Targets on the 2014 FRC Field. This article details the implementation in the LabVIEW code that matches this theoretical approach.

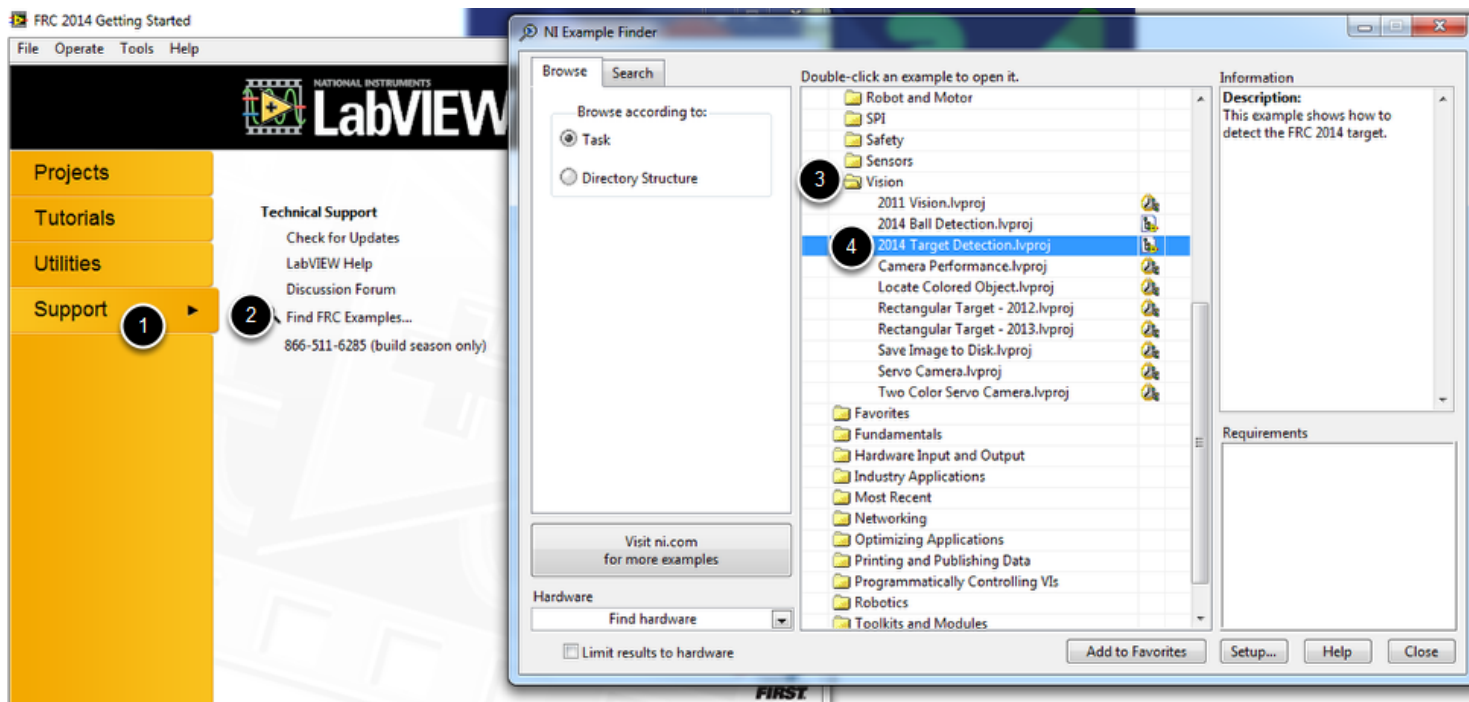
Integrating into Program



The elements of the 2014 Vision example can be incorporated into the robot program, dashboard, or contain elements in both. For more information about integrating the code into a program, see Tutorial 8 included with your LabVIEW install by clicking Tutorials -> Tutorial 8 from the Splash screen.

Vision Processing

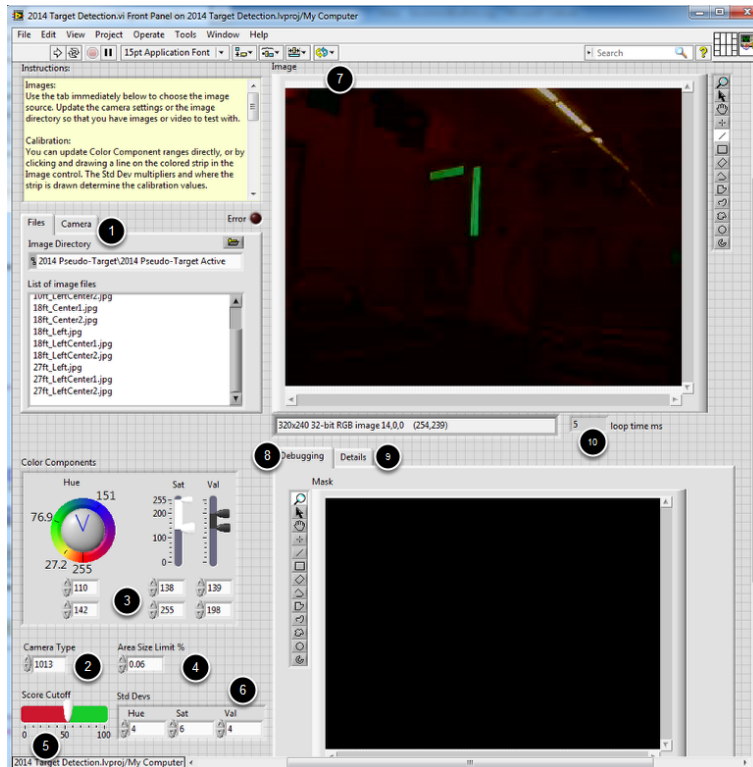
Finding the Example



The 2013 LabVIEW 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 **2014 Target Detection.lvproj**

Vision Processing

The Front Panel



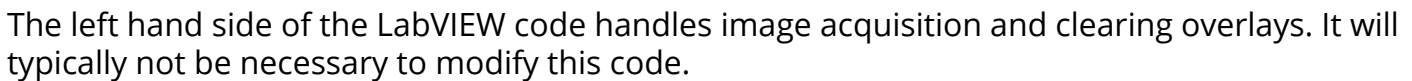
2014 Target Detection.vi is the main VI of the project. Double clicking on this VI in the Project Explorer will show the front panel shown above. The front panel contains a variety of controls and indicators used in the example

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 and dragging a line on the desired color in the Original Image section. 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"

Vision Processing

4. Area Size Limit: This controls limits the minimum particle size to be scored as a percentage of the total image size.
5. Score Cutoff: This section sets the minimum limit for each score used to determine if a target is a hot target. A target pair must exceed this minimum for the Tape Width and Vertical Score and one of the two directional scores to be considered a Hot Target.
6. Std Devs: This section is used to tweak the behavior of the automatic color selection. The numbers in the boxes represent the number of standard deviations from the mean that will be included in the range created when a line is drawn across the image.
7. Original Image: This section contains the original unprocessed image. Clicking and dragging anywhere in this image will set the color threshold values to detect that color. Detected particles will be annotated depending on whether they are determined to be a target or not. Particles which are not targets or a pair of targets which are not considered a Hot Target will be surrounded by a red box. Non-Targets will include the Aspect ratio and Rect score highlighting in red the scores which do not meet the cutoff, not-Hot pairs will include the Left, Right, Tape Width and Vertical Scores, highlighting in red scores which do not meet the cutoff. Particles which are vertical targets determined to be not hot targets are surrounded by a teal box with the Aspect Ratio and Rectangularity scores in teal. Hot targets will be surrounded by a green box with the Left, Right, Tape Width and Vertical scores in green (the unused Left or Right score will be in grey).
8. Debugging Tab: This tab shows the processed image. This is the image after thresholding, but does not show the results of the particle filtering.
9. Details Tab: Clicking on the Details tab will show an array indicator (click the arrows near the top right of the tab to view other particles) containing details on each detected particle/target.
10. Loop Time: This indicator shows the rate at which the loop is running.

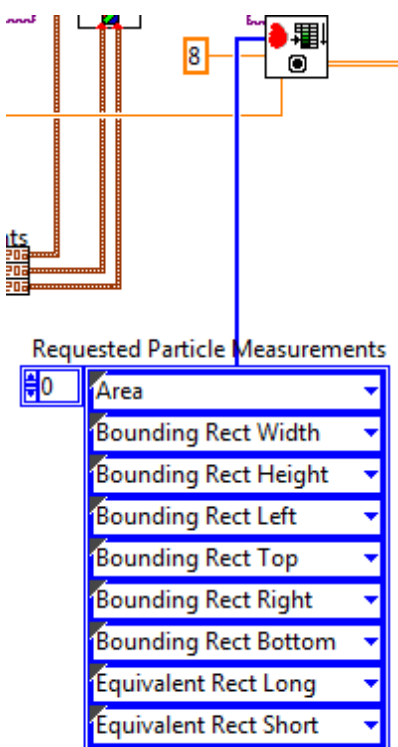
Acquiring an Image and Clearing Overlays



The top right section of the code sets the HSV threshold values on a click and drag event or passes through the values set in the controls.

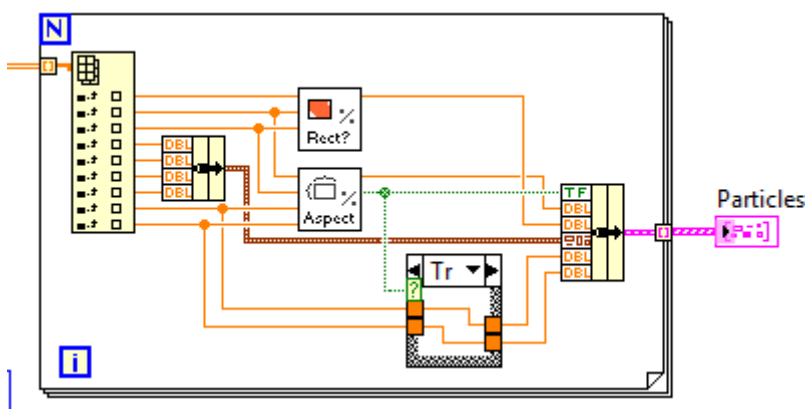
Vision Processing

Target Search Details - Particle Filter and Report



This section of the code filters out small particles and creates a report for each remaining particle (up to 8) which contains the specified measurements.

Target Search Details - Scoring

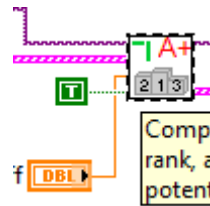


This section of the code scores the particles on Rectangularity and Aspect Ratio and indicates whether the particle is larger horizontally or vertically. It uses the orientation to pack up the

Vision Processing

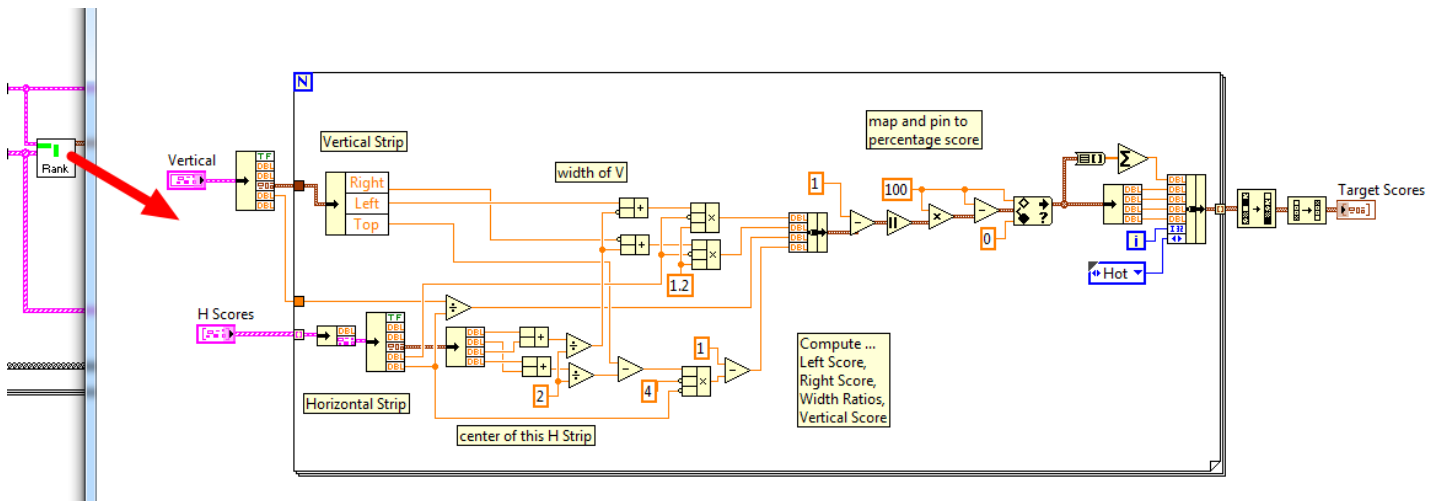
Equivalent Rectangle measurements, along with the other values, into a cluster which is output from the VI.

Score and Rank Targets



The Score and Rank Target Objects VI iterates through each vertical target and determines the best matching horizontal target then compares the Left or Right Score, Tape Width Score and Vertical Score to determine if the target is a Hot Target. This VI also annotates the image and sorts the detected targets in the order Left Targets, Right Targets, Not Hot Targets.

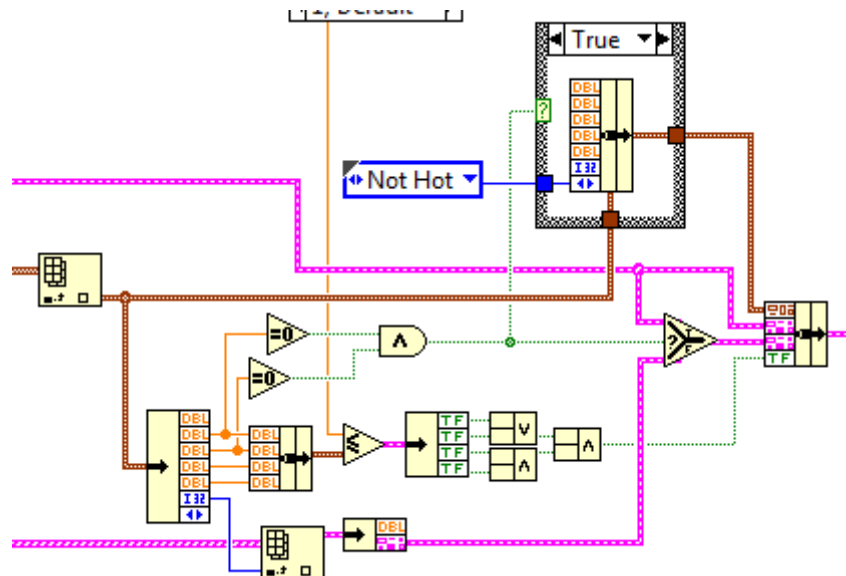
Find Horizontal



The Rank HV Combos VI iterates through each Horizontal Target and computes the Left Score, Right Score, Tape Width Score and Vertical Score. It then sorts the Horizontals by Total Score (sum of Left, Right, Width and Vertical) to select the best matching Horizontal for the Vertical.

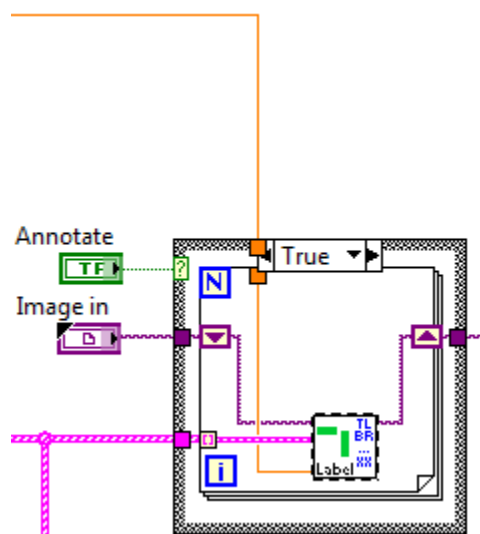
Vision Processing

Score and Rank Targets Details - Score Compare



This section of code compares the scores to the Score Limit and determines if the target is a Hot target.

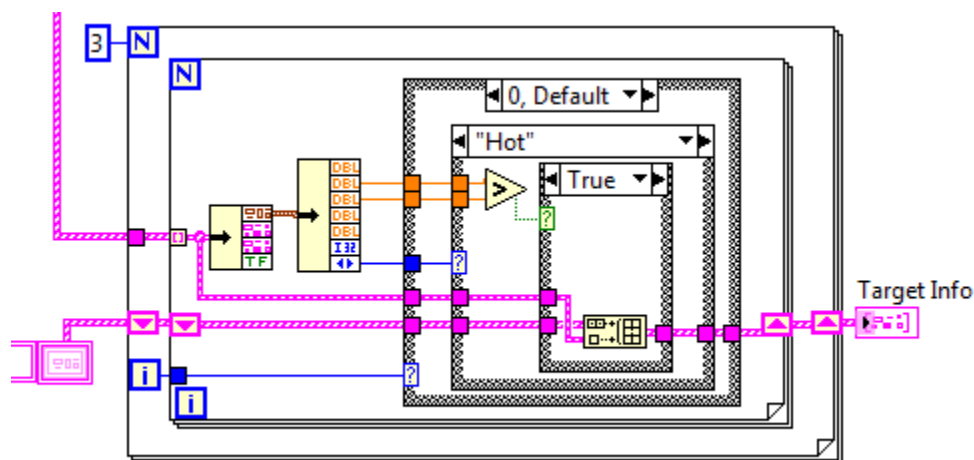
Score and Rank Target Details - Annotation



This section annotates the images with the boxes and scores for each particle.

Vision Processing

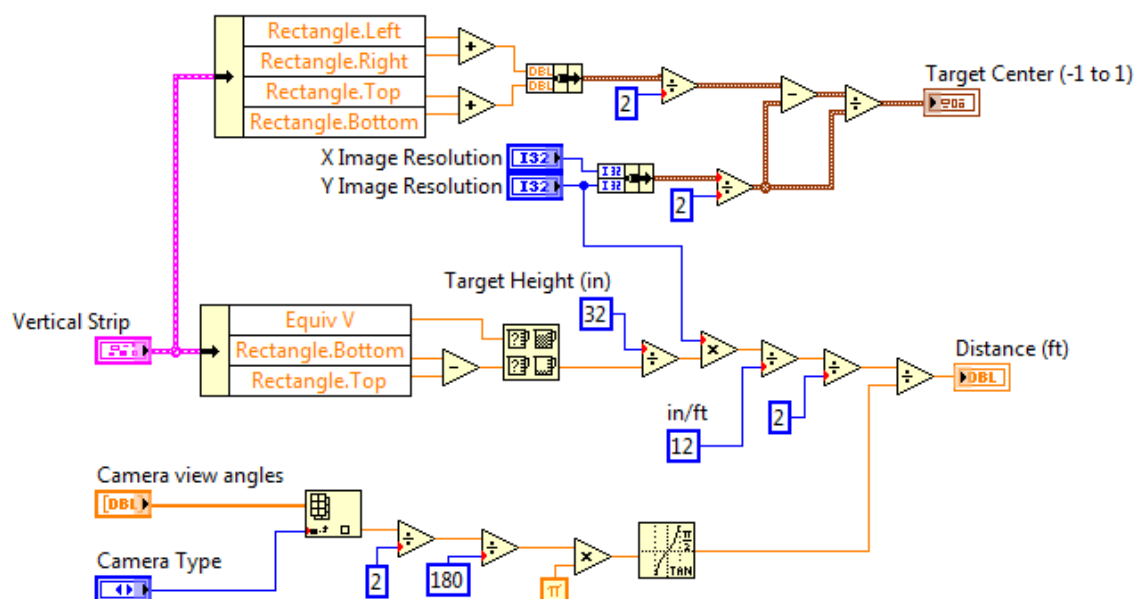
Score and Rank Target Details - Sorting



This section of the code sorts the targets placing Left Targets first in the array, then Right Targets, then Not Hot Targets

Compute Distance

Calculate the distance from center of the image and distance to wall. Documentation and explanation are shown below code



Vision Processing

The code in the Compute Target Distances 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.

Sample Images



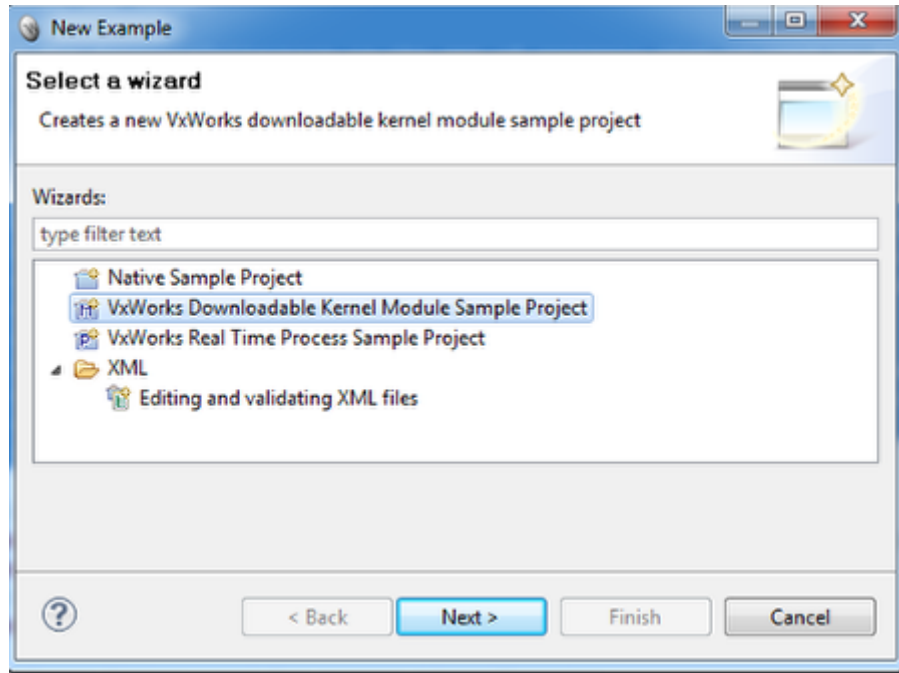
A number of sample images are provided in the example Project Directory. The provided images are broken up into groups, one group is of a pseudo-target in the Hot position (reflective tape attached directly to the polycarbonate in the correct locations). Another group is of the pseudo-target in the Not hot position. The last group is images of the actual target setup from the kickoff filming field. All lit images were taken with a pair of green LED ring lights that nest one inside the other. While these images should help teams test algorithms quickly, it is highly recommended to utilize the reflective material provided in the Kit of Parts to create a target to test the camera and lighting setup that will be used on the robot. Note that the measurements in the image filenames are very rough and should not be taken as accurate measurements to be used for distance calibration calculations.

Vision Processing

C++/Java Code

The [Identifying the Targets](#) section explains a theoretical approach to locating the Vision Targets on the 2014 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.

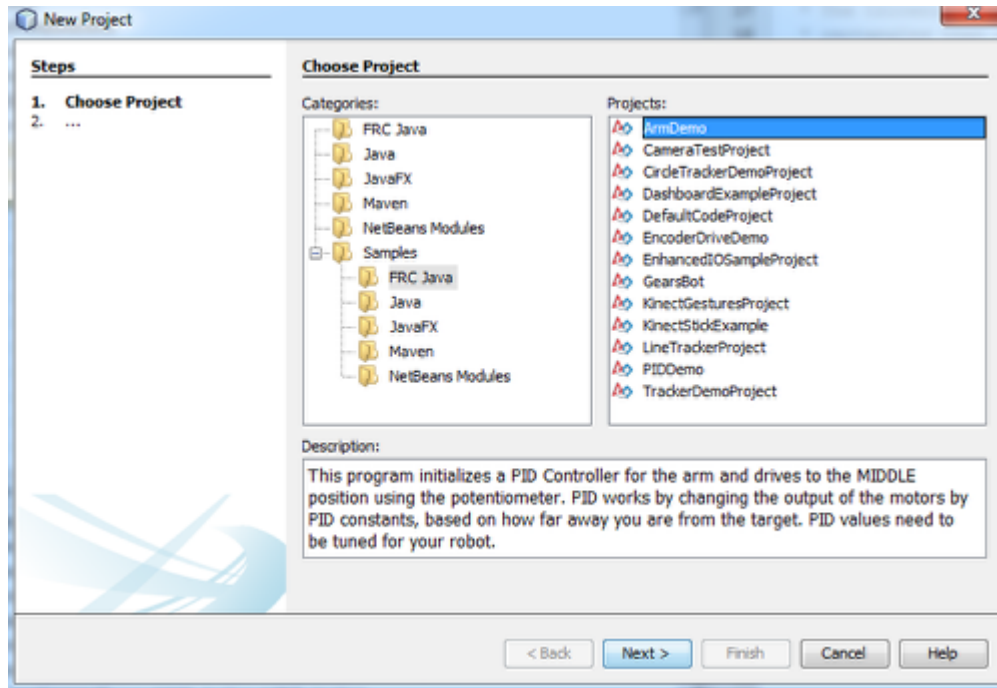
Finding the Example: C++



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 2014 Vision Sample Program and click Finish to open the sample.

Vision Processing

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 2014VisionSampleProject, 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.

Vision Processing

Code Constants

```
//Camera constants used for distance calculation
#define Y_IMAGE_RES 480      //X Image resolution in pixels, should be 120, 240 or 480
#define VIEW_ANGLE 49        //Axis M1013
// #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 40
#define ASPECT_RATIO_LIMIT 55

//Score limits used for hot target determination
#define TAPE_WIDTH_LIMIT 50
#define VERTICAL_SCORE_LIMIT 50
#define LR_SCORE_LIMIT 50

//Minimum area of particles to be considered
#define AREA_MINIMUM 150

//Maximum number of particles to process
#define MAX_PARTICLES 8
```

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 and Target Report Structures

```
//Structure to represent the scores for the various tests used for target identification
struct Scores {
    double rectangularity;
    double aspectRatioVertical;
    double aspectRatioHorizontal;
};

struct TargetReport {
    int verticalIndex;
    int horizontalIndex;
    bool Hot;
    double totalScore;
    double leftScore;
    double rightScore;
    double tapeWidthScore;
    double verticalScore;
};
```

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. A separate structure is used to contain information about targets.

Vision Processing

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, 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.

To execute properly the code, as written, must have an image named testImage.jpg stored in the cRIO's root directory. To do this 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. You can then copy and rename an image from the sample images folder described below in the "Sample Images" section. It is highly recommended to take a new set of sample images using the actual robot, camera and lighting when available.

Alternatively, commented code is also provided to read from the Axis camera.

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()];
```

Vision Processing

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

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 horizontal and vertical targets. The determination on the particle (target or not) is printed to the console along with the center and scores of the particle for debugging purposes.

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.

Score Aspect Ratio

```
double scoreAspectRatio(BinaryImage *image, ParticleAnalysisReport *report, bool vertical){
    double rectLong, rectShort, idealAspectRatio, aspectRatio;
    idealAspectRatio = vertical ? (4.0/32) : (23.5/4); //Vertical reflector 4" wide x 32" tall, horizontal 23.5" wide x 4" tall

    imaqMeasureParticle(image->GetImaqImage(), report->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_LONG_SIDE, &rectLong);
    imaqMeasureParticle(image->GetImaqImage(), report->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_SHORT_SIDE, &rectShort);

    //Divide width by height to measure aspect ratio
    if(report->boundingRect.width > report->boundingRect.height){
        //particle is wider than it is tall, divide long by short
        aspectRatio = ratioToScore(((rectLong/rectShort)/idealAspectRatio));
    } else {
        //particle is taller than it is wide, divide short by long
        aspectRatio = ratioToScore(((rectShort/rectLong)/idealAspectRatio));
    }
    return aspectRatio; //force to be in range 0-100
}
```


Vision Processing

The scoring of the Aspect Ratio is broken out into it's own method for clarity. This method compares the aspect ratio of the equivalent rectangle to the ideal aspect ratio for the target (which target type to use is determined by a parameter passed in)

Score Rectangularity

```
double scoreRectangularity(ParticleAnalysisReport *report){
    if(report->boundingRect.width*report->boundingRect.height !=0){
        return 100*report->particleArea/(report->boundingRect.width*report->boundingRect.height);
    } else {
        return 0;
    }
}
```

The scoring of the Aspect Ratio is broken out into it's own method for clarity. This method compares the area of the particle to the area of the bounding box and returns a score between 0 and 100.

Ratio To Score

```
double ratioToScore(double ratio)
{
    return (max(0, min(100*(1-fabs(1-ratio)), 100)));
}
```

Many of the score calculations utilize the same subcalculation to convert a ratio with an ideal value of 1 to a 0-100 score value using a piecewise linear function that goes from (0,0) to (1,100) to (2,0). This calculation was broken out into a method to allow the code to be re-used for multiple score calculations.

Score Compare

```
bool scoreCompare(Scores scores, bool vertical){
    bool isTarget = true;

    isTarget &= scores.rectangularity > RECTANGULARITY_LIMIT;
    if(vertical){
        isTarget &= scores.aspectRatioVertical > ASPECT_RATIO_LIMIT;
    } else {
        isTarget &= scores.aspectRatioHorizontal > ASPECT_RATIO_LIMIT;
    }

    return isTarget;
}
```

The scoreCompare method checks if a particle is a target (meets all of the score minimums) of a specified type.

Vision Processing

Finding Hot Targets

```
//Zero out scores and set verticalIndex to first target in case there are no horizontal targets
target.totalScore = target.leftScore = target.rightScore = target.tapeWidthScore = target.verticalScore = 0;
target.verticalIndex = verticalTargets[0];
for (int i = 0; i < verticalTargetCount; i++)
{
    ParticleAnalysisReport *verticalReport = &(reports->at(verticalTargets[i]));
    for (int j = 0; j < horizontalTargetCount; j++)
    {
        ParticleAnalysisReport *horizontalReport = &(reports->at(horizontalTargets[j]));
        double horizWidth, horizHeight, vertWidth, leftScore, rightScore, tapeWidthScore, verticalScore, total;

        //Measure equivalent rectangle sides for use in score calculation
        imgMeasureParticle(filteredImage->GetImagImage(), horizontalReport->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_LONG_SIDE, &horizWidth);
        imgMeasureParticle(filteredImage->GetImagImage(), verticalReport->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_SHORT_SIDE, &vertWidth);
        imgMeasureParticle(filteredImage->GetImagImage(), horizontalReport->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_SHORT_SIDE, &horizHeight);

        //Determine if the horizontal target is in the expected location to the left of the vertical target
        leftScore = ratioToScore(1.2*(verticalReport->boundingRect.left - horizontalReport->center_mass_x)/horizWidth);
        //Determine if the horizontal target is in the expected location to the right of the vertical target
        rightScore = ratioToScore(1.2*(horizontalReport->center_mass_x - verticalReport->boundingRect.left - verticalReport->boundingRect.width)/horizWidth);
        //Determine if the width of the tape on the two targets appears to be the same
        tapeWidthScore = ratioToScore(vertWidth/horizHeight);
        //Determine if the vertical location of the horizontal target appears to be correct
        verticalScore = ratioToScore(1-(verticalReport->boundingRect.top - horizontalReport->center_mass_y)/(4*horizHeight));
        total = leftScore > rightScore ? leftScore:rightScore;
        total += tapeWidthScore + verticalScore;

        //If the target is the best detected so far store the information about it
        if (total > target.totalScore)
        {
            target.horizontalIndex = horizontalTargets[j];
            target.verticalIndex = verticalTargets[i];
            target.totalScore = total;
            target.leftScore = leftScore;
            target.rightScore = rightScore;
            target.tapeWidthScore = tapeWidthScore;
            target.verticalScore = verticalScore;
        }
    }
}
//Determine if the best target is a Hot target
target.Hot = hotOrNot(target);
```

This section of the code iterates through each previously detected vertical target and calculates the scores for each detected horizontal target. After a pair is scored, the code checks if the total score for this pair is the highest detected so far; if so, information about the target is saved for later use. Finally the code checks if the target is a hot target or not.

Hot or Not

```
bool hotOrNot(TargetReport target)
{
    bool isHot = true;

    isHot &= target.tapeWidthScore >= TAPE_WIDTH_LIMIT;
    isHot &= target.verticalScore >= VERTICAL_SCORE_LIMIT;
    isHot &= (target.leftScore > LR_SCORE_LIMIT) | (target.rightScore > LR_SCORE_LIMIT);

    return isHot;
}
```

The hotOrNot method compares the scores for a target to the specified minimums in order to determine if the target is a Hot Target.

Vision Processing

Print Target Info

```
if(verticalTargetCount > 0)
{
    //Information about the target is contained in the "target" structure
    //To get measurement information such as sizes or locations use the
    //horizontal or vertical index to get the particle report as shown below
    ParticleAnalysisReport *distanceReport = &(reports->at(target.verticalIndex));
    double distance = computeDistance(filteredImage, distanceReport);
    if(target.Hot)
    {
        printf("Hot target located \n");
        printf("Distance: %f \n", distance);
    } else {
        printf("No hot target present \n");
        printf("Distance: %f \n", distance);
    }
}
```

This section of the code computes the distance to the best detected target and prints out if the best target is a Hot target or not and the distance to the target.

Computing Distance

```
double computeDistance (BinaryImage *image, ParticleAnalysisReport *report) {
    double rectLong, height;
    int targetHeight;

    imaqMeasureParticle(image->GetImaqImage(), report->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_LONG_SIDE, &rectLong);
    //using the smaller of the estimated rectangle long side and the bounding rectangle height results in better performance
    //on skewed rectangles
    height = min(report->boundingRect.height, rectLong);
    targetHeight = 32;

    return Y_IMAGE_RES * targetHeight / (height * 12 * 2 * tan(VIEW_ANGLE*PI/(180*2)));
}
```

The computeDistance method computes the distance to a vertical target using the approach described in the previous article. The image must be passed to the method so the method can make the equivalent rect measurement.

Cleanup

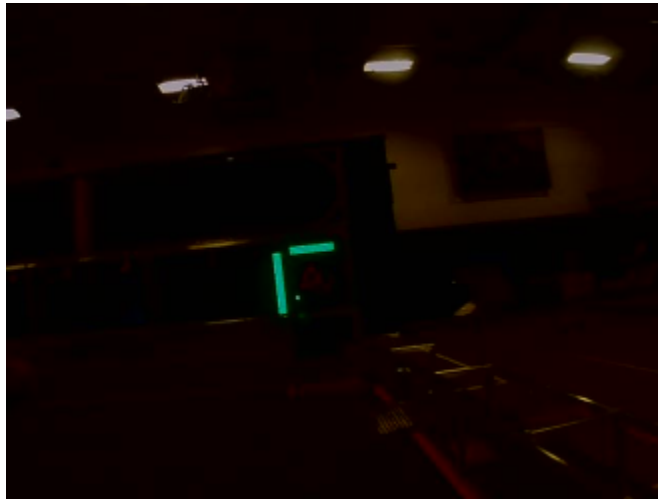
```
// be sure to delete images after using them
delete filteredImage;
delete thresholdImage;
delete image;

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

Vision Processing

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 are provided in the VisionImages folder in the example Project Directory. The provided images are broken up into groups, one group is of a pseudo-target in the Hot position (reflective tape attached directly to the polycarbonate in the correct locations). Another group is of the pseudo-target in the Not hot position. The last group is images of the actual target setup from the kickoff filming field. All lit images were taken with a pair of green LED ring lights that nest one inside the other. While these images should help teams test algorithms quickly, it is highly recommended to utilize the reflective material provided in the Kit of Parts to create a target to test the camera and lighting setup that will be used on the robot. Note that the measurements in the image filenames are very rough and should not be taken as accurate measurements to be used for distance calibration calculations.

Axis M1013 Camera Compatibility

It has come to our attention that the Axis M1011 camera has been discontinued and superseded by the Axis M1013 camera. This document details any differences or issues we are aware of between the two cameras when used with WPILib and the provided sample vision programs.

Optical Differences

The Axis M1013 camera has a few major optical differences from the M1011 camera:

1. The M1013 is an adjustable focus camera. Make sure to focus your M1013 camera by turning the grey and black lens housing to make sure you have a clear image at your desired viewing distance.
2. The M1013 has a wider view angle (67 degrees) compared to the M1011 (47 degrees). This means that for a feature of a fixed size, such as the 4 in. wide retroreflective tape, the image of that feature will span a smaller number of pixels

Using the M1013 With WPILib

The M1013 camera has been tested with all of the available WPILib parameters and the following performance exceptions were noted:

1. The M1013 does not support the 160x120 resolution. Requesting a stream of this resolution will result in no images being returned or displayed.
2. The M1013 does not appear to work with the Color Enable parameter exposed by WPILib. Regardless of the setting of this parameter a full color image was returned.

All other WPILib camera parameters worked as expected. If any issues not noted here are discovered, please file a bug report on the [WPILib tracker](#) (note that you will need to create a FIRSTForge account if you do not have one, but you do not need to be a member of the project).

Using the M1013 With Vision Sample Code

If using the M1013 camera with the provided Vision sample code, or code based on it, please note the following:

1. Per #2 in Optical Differences above, the view angle of the camera is different than the M1011. The code for distance calculation will need to be modified to use the appropriate angle. Based on testing with the 206 and M1011 cameras, teams may wish

Vision Processing

to take measurements of the actual distance to the target and corresponding image parameters and calculate an empirical view angle.

2. Per #2 in Using the M1013 With WPILib above, the Color Enable parameter does not appear to work with the M1013. This does **not** affect the Monochrome option of the LabVIEW sample. This option of the sample code processes a single plane of the image, but does not request a monochrome image.