

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

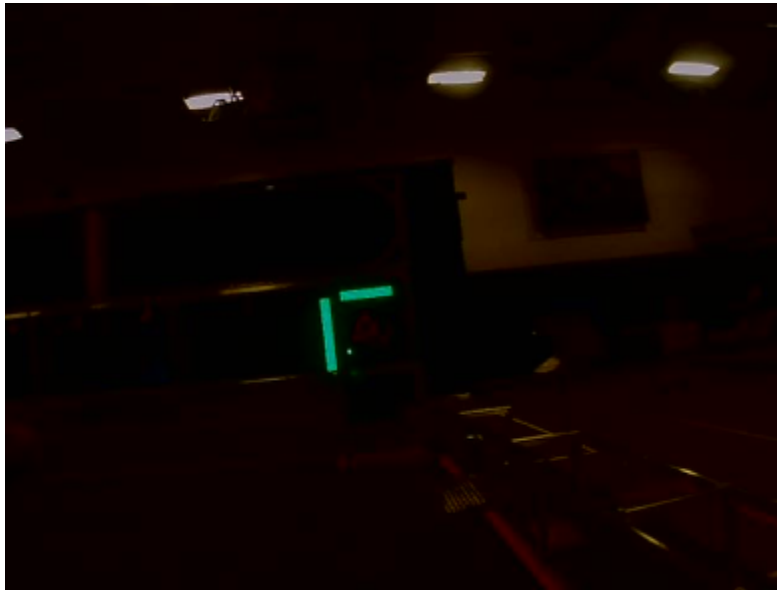
This document walks through the approach used by the example code provided in LabVIEW (for PC or cRIO), C++ and Java, details on the language specific examples are provided in subsequent articles. In addition to these options teams should be aware of the following alternatives that allow for vision processing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

Identifying and Processing the Targets



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.

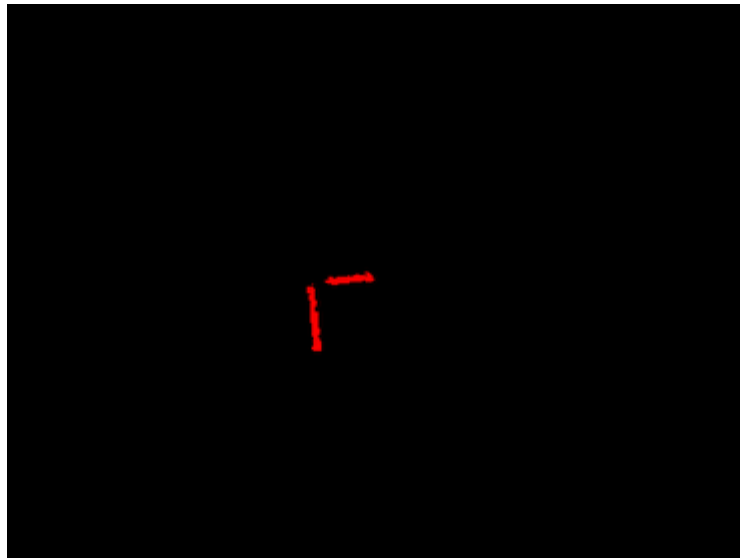
The example code uses the HSV color space to specify the color of the target. The primary reason is that it readily allows for using the brightness of the targets relative to the rest of the image as a filtering criteria by using the Value (HSV) or Luminance (HSL) component. Another reason to use the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

Identifying and Processing the Targets

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.



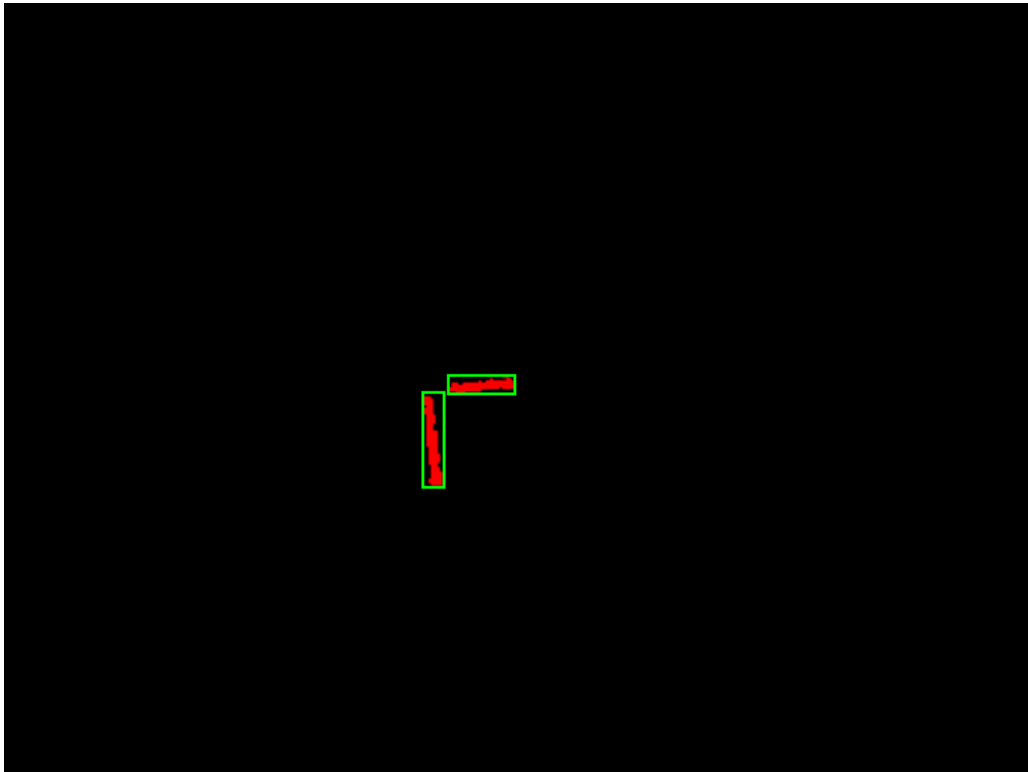
Particle Analysis

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.

Identifying and Processing the Targets



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.

Identifying and Processing the Targets

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

Identifying and Processing the Targets

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.

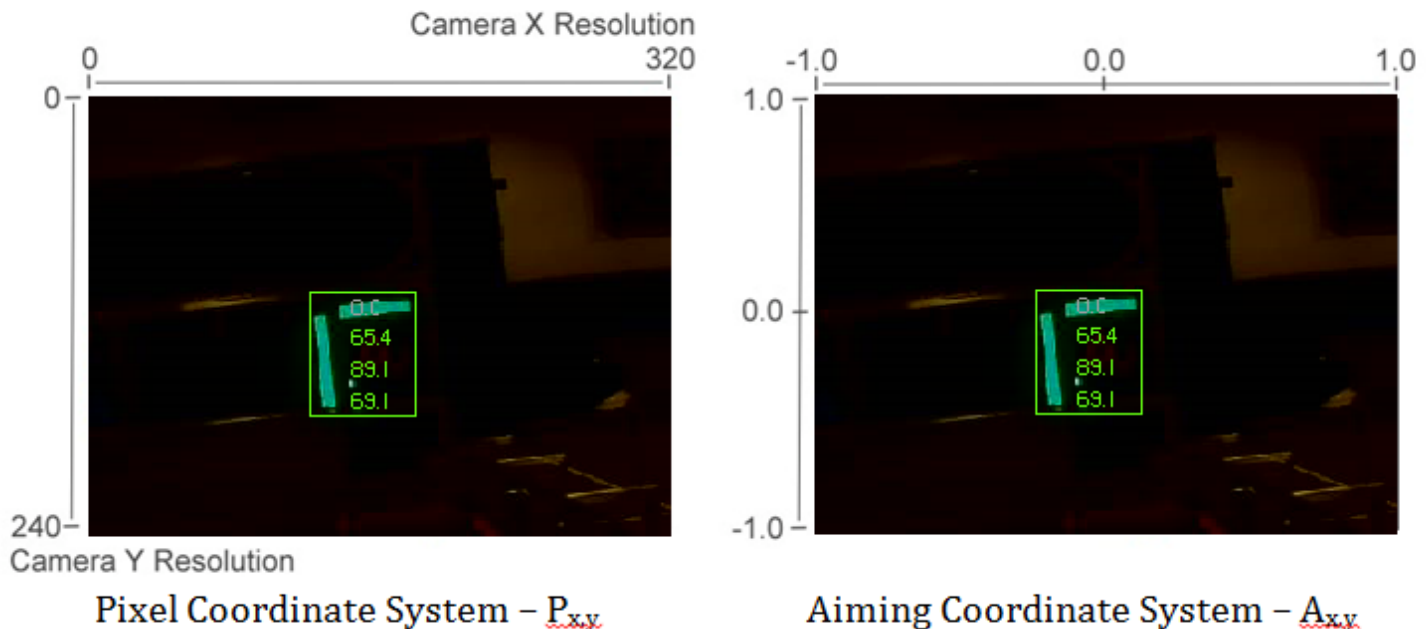
To convert a point from the pixel system to the aiming system, we can use the formula shown below.

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

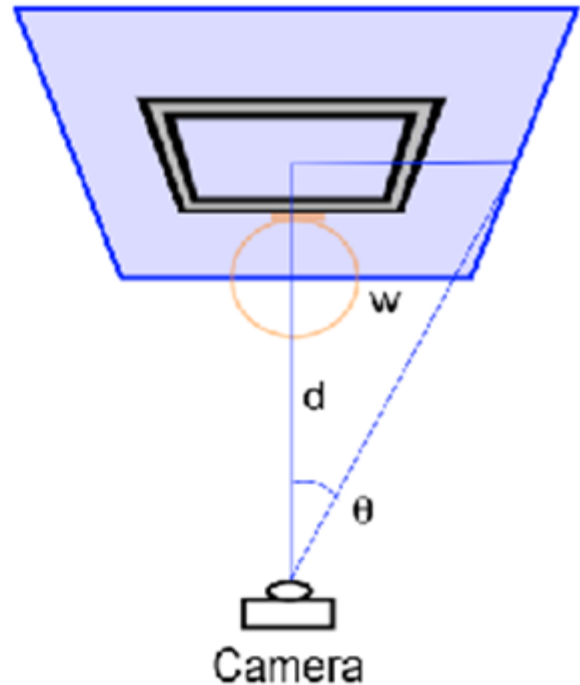
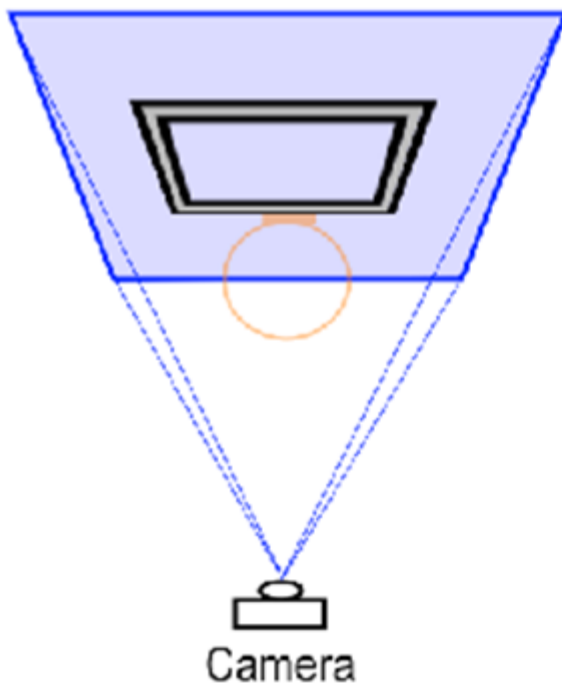
The target distance is computed with knowledge about the target size and the camera optics. The approach uses information about the camera lens view angle and the width of the camera field of

Identifying and Processing the Targets

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 ...

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

Identifying and Processing the Targets

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.