Jack LeFevre
4/6/2017

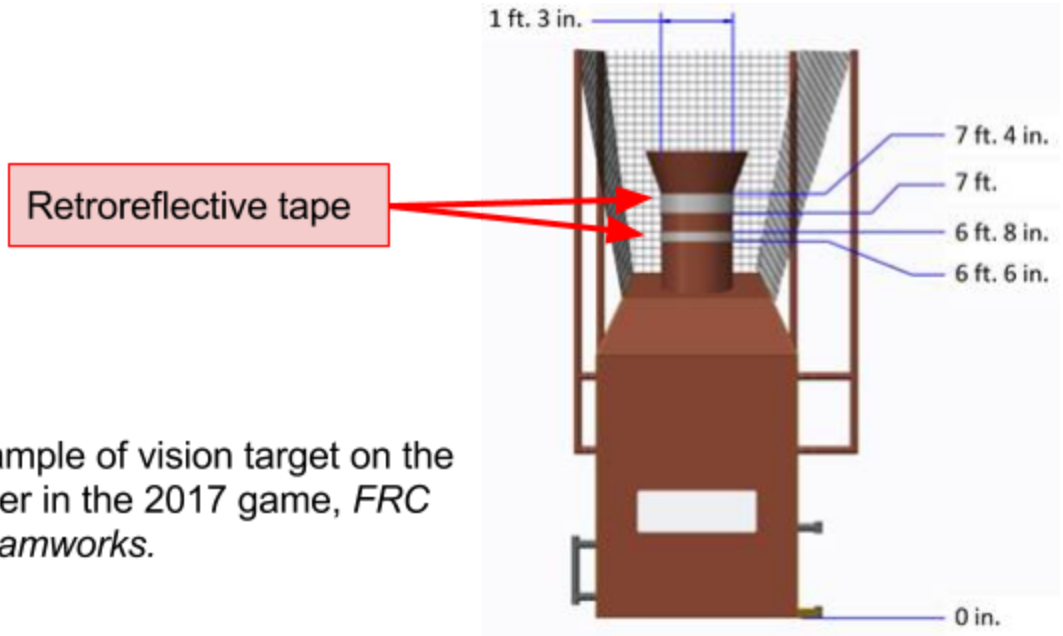
Vision Targeting

This white paper describes the uses of vision targeting for FRC games, as well as how vision targeting works.

In this paper, I will be using the example of programming a vision coprocessor (Like a Raspberry Pi) with Python and OpenCV installed.

Vision Targets

FRC uses vision targets on various field elements to help out teams using vision targeting. A vision target consists of retroreflective tape in various patterns on the field elements.



Example of vision target on the boiler in the 2017 game, *FRC Steamworks.*

Retroreflective tape has a special type of reflective surface which reflects light directly back where it came from, regardless of its angle. This can be demonstrated by placing

a piece of the tape across the room, and shining a flashlight at it while gradually moving the flashlight towards your eyes. When the flashlight is close to your eye, the tape will appear to glow brightly.

This retroreflective property is similar to the reflectors on road signs and on some bicycles, which are used for night-time visibility.

Vision Targeting

The vision targets can be used for vision targeting, in which a camera mounted on a robot captures an image and processes it to find where or how far away a target is, then acts on those measurements autonomously. To achieve this, a robot must go through several steps:

1. Illuminate the target

The vision targets don't stand out on their own, which makes finding them in an image difficult. However, since the targets are made of retroreflective tape, an easy way to make them stand out is to shine light on the target with a light mounted near the camera.

Since most buildings where events will be held have white-colored ceiling lights, it is not wise to use a white light to illuminate the target. Instead, it is best to use green or blue lights.

Most teams use LED ring lights around their camera, since they provide the most light closest to the camera. Ring lights like this can be found here.

2. Capture and threshold an image

Once the target is lit up, an image must be captured. Since the retroreflective tape will appear to glow brightly, it is best to capture the image with a very low exposure setting on the camera.

Since the retroreflective tape will appear to glow brightly within the image, it is possible to threshold the image to filter out everything else. Thresholding is the process of creating a black-and white 'mask' from an image by setting pixels to white where the original image is within a certain threshold, and setting the pixels black elsewhere.

To threshold the image, several things need to be done. First, the image should be converted from the RGB (Red-Green-Blue) format which the camera provides to HSB (Hue-Saturation-Brightness). This allows much easier filtering.

Next, an acceptable color range should be found. This can be done by altering the values of the threshold until only the vision target appears. It is important to set the threshold so that it works at various distances and angles to ensure that the target can always be found. However, it is also important to filter out any large background areas that are found. Small flickers are easy to ignore, but larger areas can be easy to confuse with the target.

Finally, the image must be thresholded. In Python with OpenCV, this is done with the "InRange" function, which takes the minimum/maximum HSV values and the input image and returns a black-and-white image which highlights only the areas of the

correct color. This image is called a ‘mask,’ since it can be used to mask out regions of the image in which the color is not correct.

An example of a thresholding operation applied to an image of a hand

3. Find contours in the image

Once the image is thresholded to find the areas with vision tape, the target must be processed to be in a usable format for the robot. The way this is done is by finding contours in the mask.

Contours are polygons which represent the outline of white areas in the mask. With contours, it is possible to find the corners, center, size, shape, etc. of an object.

Finding contours is a simple operation with OpenCV and Python. Contours can be found by using the “findContours” function.

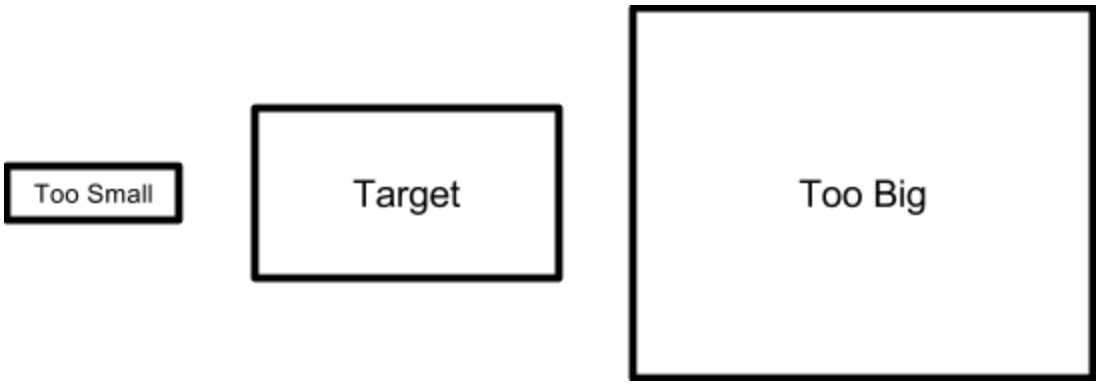
4. Filtering contours

When contours are found in the image, there are typically some which are not part of the target. To get meaningful information about the target’s position, it is necessary to filter out and ignore the contours which are not part of the target. This can be done by finding the features of the target and looking for contours with similar features.

Shape features:

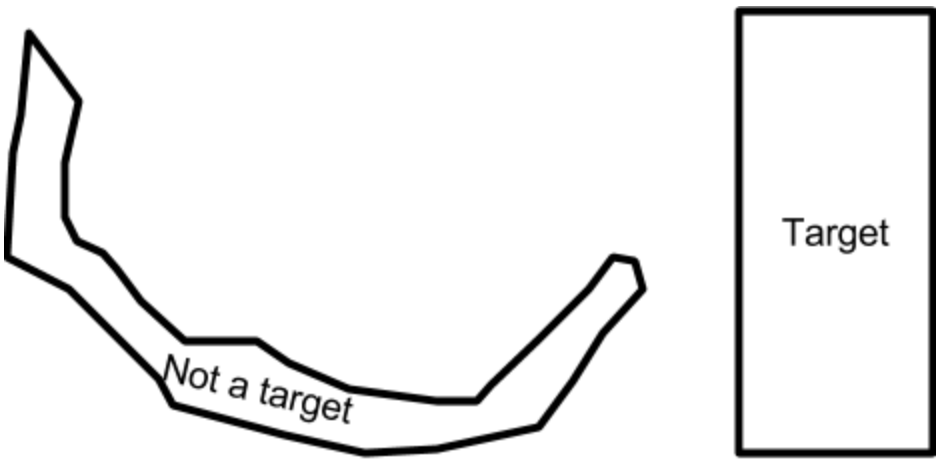
Since the shape of the target is well-known, it is easy to filter out contours which are not of the same shape.

The easiest way to do this is to find contours with the correct size. This can filter out small flecks in the image as well as large objects like the ceiling in the background.



Another way is to look at the aspect ratio of the contour, which is the width of the target divided by the height of the target. If the target is a tall, skinny rectangle, then short, wide rectangles will be filtered out.

A final way to filter based on shape is to look at how convex the contour is. Often times, a contour that is not the target will be a large, skinny ring or curve. While this may have close to the correct shape, size, and area, it wouldn't be convex. These kinds of shapes can be filtered out by finding the area of their convex hull and comparing it to their actual area. If there is a large difference, the shape is likely not a target.



Location features:

Another feature distinguishing targets from non-targets is their location. Often, the vision targets are made up of two pieces, either on top of each other or side by side. These targets can be found by looking for two contours that meet the other criteria and are at the same X position for vertical targets and the same Y position for horizontal targets.

Other targets are near ground level. When looking for one of these targets, the other contours can be filtered out if they are too high up.

After the contours have been filtered out, only one or two should be left, depending on whether the target has one or two pieces of tape.

5. Measuring the target and communicating with the robot

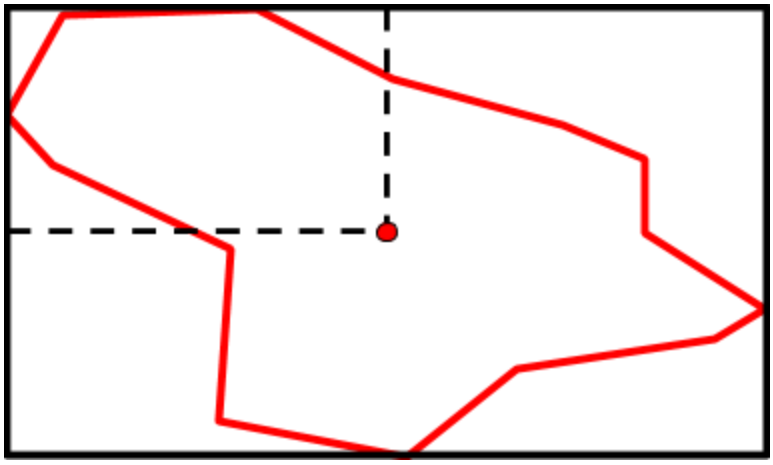
After the target has been found, useful information about it can be calculated. This includes its position, size, and distance.

Position:

The target's position can be determined in several ways: Finding the center of the bounding rectangle, finding the center of the minimum area rectangle, or finding the centroid of the contour.

The simplest method for determining the position of a contour is to use its bounding rectangle. The bounding rectangle is the smallest rectangle that can fit around the

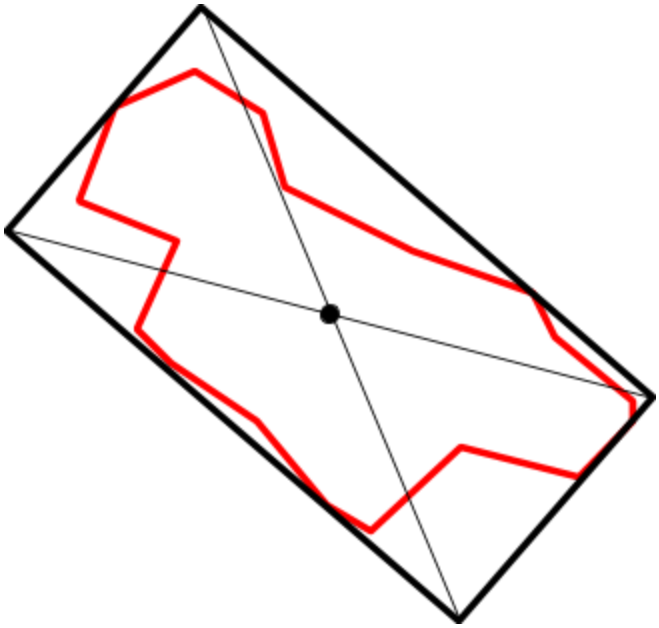
contour without rotating. This method works best for rectangular targets that are oriented vertically. The bounding rectangle can be found with the OpenCV function 'boundingRect()'. Once the bounding rectangle is found, the center can be found by adding half the width to the X position and half the height to the Y position.



Finding the center of the bounding rectangle

The second way to find the center of the contour is to find the center of the minimum area rectangle around the contour. This method is ideal for rectangular targets that aren't always oriented vertically.

The minimum area rectangle can be calculated by using the OpenCV function 'minAreaRect()'. This function returns a variable representing the position and rotation of the rectangle, which makes it difficult to find the center. Instead, using the 'boxPoints()' function on the output gives the coordinates of the four corners of the rectangle. This makes it easy to calculate the center point of the rectangle by averaging the coordinates of each of the four points.



A contour with the center found using the minimum area rectangle.

The final method to find the position of a contour is to find the centroid, or the center of gravity. The centroid is defined as the average position of all of the points in the shape. The centroid is useful for contours with irregular or non-rectangular shapes.

The easiest way to find the centroid of a contour is to use the image moments, which are numbers that describe the contour and make it easy to calculate various measurements about the contour. The image moments can be calculated using the OpenCV 'moments()' function, which returns 'Moments' structure which contains various numbers, like m00, m01, and m10, which are used to find the centroid. With moments M, the centroid (cx and cy) can be found as follows:

```
cx = int(M['m10']/M['m00'])
cy = int(M['m01']/M['m00'])
```

Size:

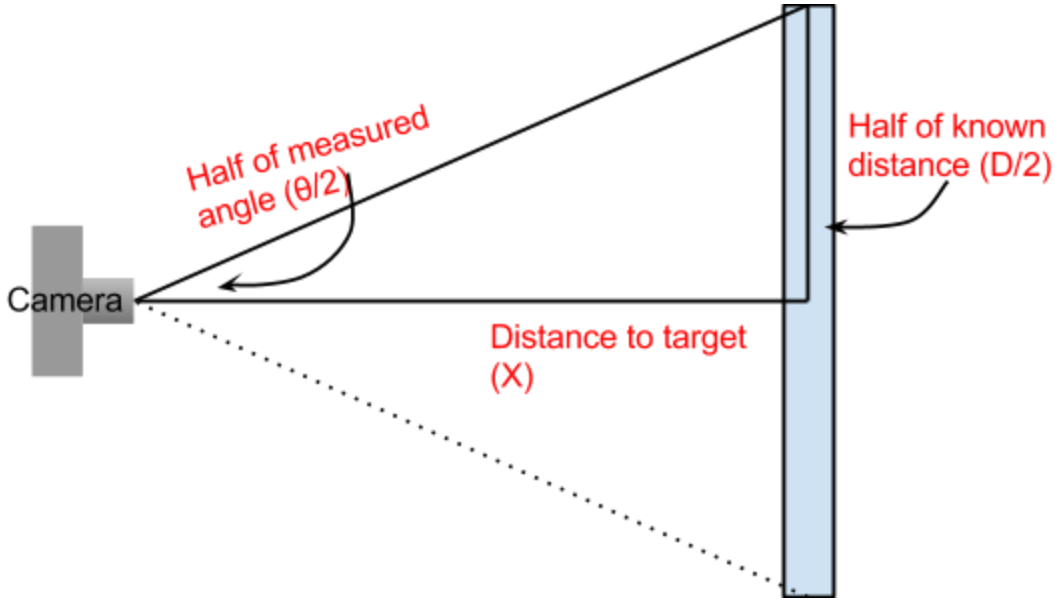
Finding the size of a contour is very similar to finding the position of the contour. Using either the bounding rectangle or the minimum area rectangle, the size is simply either the width or height of the rectangle, depending on which measurement is needed. Calculating the size can be very useful in determining a target's distance.

Distance:

Finding the distance to a target relies on knowing two things: The conversion from pixels to degrees in an image (Which is discussed below), and the actual size of a target.

To determine the distance to a target, the target is first measured in the image. This can be done by determining the size of the target as described above, or by determining the distance between two parts of the target for two-piece targets. This measurement must be converted to degrees, then can be plugged into an equation to find the distance.

Since a triangle can be made from the width of the target and the angle from the image, it is possible to solve for the distance using trigonometry.



$tan(\theta/2) = \frac{(D/2)}{X}$, so the distance can be found by $X = \frac{D}{2tan(\theta/2)}$.

Converting pixels to degrees:

To convert pixels to degrees, the first thing needed is the total field of view of the camera which is being used. This can be found on the camera's datasheet. For example, the Microsoft LifeCam hd-3000 cameras used on the 2016 and 2017 robots have a diagonal field of view of 68.5°.

Next, the number of degrees per pixel need to be calculated. To do this, the diagonal length in pixels of an image is needed, which can be found using the Pythagorean theorem. For a resolution of 640x480 pixels, the diagonal distance in degrees is $\sqrt{640^2 + 480^2} = 800$ pixels, which means there is 0.085625 degrees per pixel. Multiplying any number of pixels by the degrees per pixel constant will give the number of degrees.

Why use vision targeting?

Vision targeting is not always the way to go, although it has many advantages over manual driving. Automatic vision targeting won't get stressed or react to the pressure in the heat of a match, and it has a point of view on the field that the drivers don't have, which makes it ideal for precise alignment with game elements. Vision targeting is also far faster than a human driver: Vision targeting code can find a target and send it to the roborio within just a few milliseconds to align the robot. However, vision targeting is far more susceptible to changing field conditions, like moving robots in front of the camera.

In situations like aligning to shoot at a target or driving precisely to align with a field element, vision targeting is clearly the best option for consistency, speed, and accuracy.

Works Cited

"Be Awesome at Learning OpenCV, Python, and Computer Vision." *PyImageSearch*. Web. 06 Apr. 2017.

"OpenCV-Python Tutorials." *OpenCV-Python Tutorials — OpenCV 3.0.0-dev Documentation*. Web. 06 Apr. 2017.

“Opencv Documentation." *Opencv Documentation — OpenCV 2.4.13.2 Documentation*.

Web. 06 Apr. 2017.