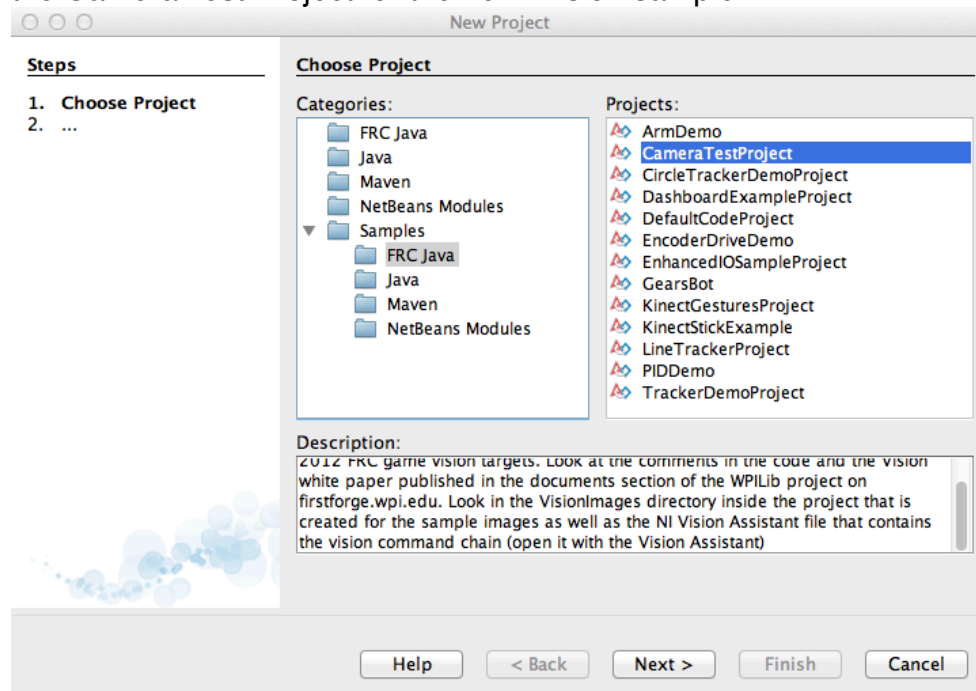


## Java Sample

The sample Java Vision program demonstrates using some of the NIVision library functions to find the 4 rectangular targets that are part of the 2012 FRC game challenge. The sample program includes a number of still photos of the target that can be used with the Vision Assistant to adjust the parameters used for the NI Vision functions used in this example. The images can also be downloaded with FTP to the cRIO to test the actual program at your desk without having to set up vision targets. In addition a Vision Assistant save file can also be found in the test projects directory and can be opened to see the chain of vision operations that were used to make this program.

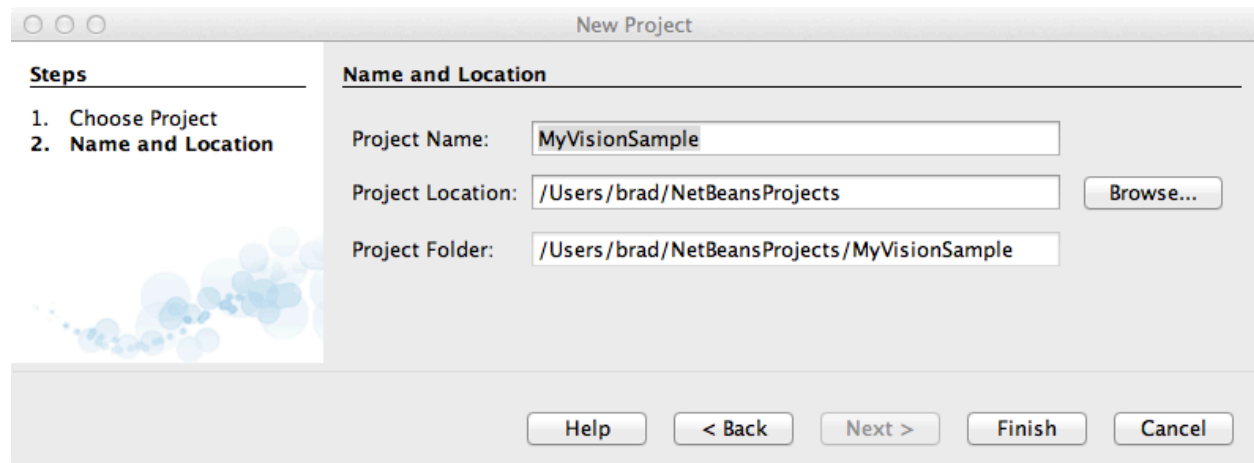
### Installing the sample program

You can get the sample program in Netbeans by selecting “File/New Project...”. Choose the Camera Test Project for the 2012 vision sample.

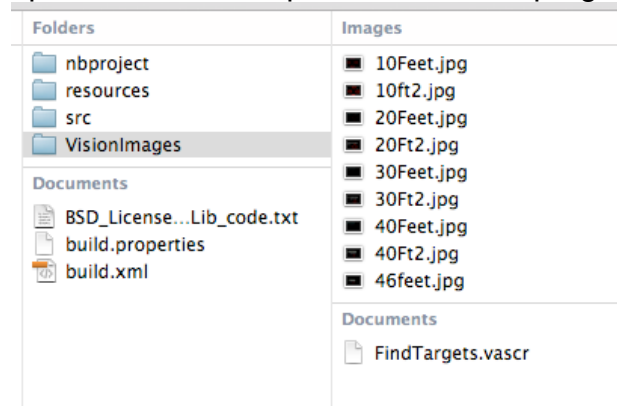


Fill in the project name, location, and folder fields and click “Finish”.

## Using the Java Vision Sample Program



Inside the project directory you'll see a folder called VisionImages that contains a number of sample photos as well as the Vision Assistant project file that shows the operations that are performed in this program.



### Downloading an image to the cRIO

The sample program can either use the camera for capturing images or by opening a representative sample image stored on the cRIO. The image it uses (as programmed) is called "10ft2.jpg" and should be downloaded using FTP to the root directory of the cRIO. By default the cRIO ftp server has a username of "root" and no password.

## Creating a vision program that runs on the cRO

The steps used in creating this program (and can be applied to any program) were:

1. Set up the targets in our lab and take sample representative photos from various distances and angles using the intended camera settings.
2. Load these images into the NI Vision Assistant to use as samples when evaluating the results of the algorithm developed.
3. Create an algorithm to find the targets that is as few steps as possible and succeeds in isolating each of the targets in each of the sample images.
4. Translate the algorithm into either C++ or Java code taking the parameters from each of the blocks in the Vision Assistant project.
5. Use the results from the code to allow the robot to track to the targets as required by your teams goals and mechanisms on your robot.

### Capturing sample images

Having representative images of the target makes it possible to experiment with your vision algorithm without having to set up the robot and do complex debugging operations. The program can then be tuned using each of the samples to make sure it can find the target in all the expected conditions.

You should take a series of photos of the target using the Axis camera at the height it will be on your robot and at the distance from the target you intend to use it. If you plan on shooting from a number of places on the field, then a number of representative images will ensure that your program can handle all the cases. You can then apply your algorithm to each of these samples and immediately see the results before you build a program for your robot. It is important to use the same camera settings as will be used on the robot. If there are number of settings that you would like to try, take photos using each of them. If you are planning on putting a ring light on the camera, be sure to do that for the sample images.

The sample project includes a number of images you can look at to get started before you take your own pictures. Create the sample project as described below to get the sample images and the Vision Assistant project file. These images were taken using the standard 2012 Axis camera that is included in the rookie kits. It was enhanced with a pair of ring lights available from FIRSTChoice. Because the retroreflective tape on the targets reflects the light coming from the same axis as the camera lens, the tape is especially bright. The exposure on the camera was set by intentionally overexposing the image by shining a flashlight into the camera, allowing the auto exposure to reduce the sensitivity, then locking in that setting. The result is that the only things visible in the images are the most brightly illuminated objects in the scene: the retroreflective tape on the targets and the room lights.

### Load the images into the Vision Assistant

#### Create an algorithm to find the targets

This sample Vision Assistant sample project (in the VisionImages subdirectory in the project directory) represents one way of finding the targets that was developed in a short period of time and designed to be very easy to understand. You can use this as a starting point, but it is recommended that you understand the program and enhance it

work for your application. There are likely optimizations that can be performed to get a higher frame rate from the camera as well as better accuracy.

The operations performed in this project consisted of the following:

thresholdRGB	Keeps only the part of the image that matches the color parameters specified by the parameters. They represent the low and high red, green and blue values in the image. This was based on the red colored LEDs used to illuminate the targets in this example. You should take your own sample photos and experiment with them.
removeSmallObjects	This removes small objects from the images. In the samples there were a number of images with a lot of reflected light that caused many small colored areas that were not part of any of the larger target segments.
convexHull	This fills in the rectangles as if a rubber band was stretched around each of them. This makes locating the rectangles in the next step much easier.
particleFilter	This step finds the rectangles in the scene and removes the ones that don't match the specified criteria, in this case a width and height of 30-400 and 40-400 pixels. These numbers were selected based on the sample images.
getOrderedParticleAnalysisReports	returns the information for each of the rectangles found in the previous step. Each particle analysis report object contains a number of parameters for the associated rectangle.

From this information your program can control the robot to track to the desired target.

### **Translating the Vision Assistant project to code**

The test program consists of one or more lines of code for each of the blocks in the Vision Assistant project. It will display each of the targets that were found in the image and one of the parameters (center of mass x value) for that target. The rest of the results for the located rectangles are stored in the ParticleAnalysisReport objects.

The following lines of code actually process the image and generate the results:

```
ColorImage image;  
image = new RGBImage("/10ft2.jpg");  
BinaryImage thresholdImage = image.thresholdRGB(25, 255, 0, 45, 0, 47);  
BinaryImage bigObjectsImage = thresholdImage.removeSmallObjects(false, 2);  
BinaryImage convexHullImage = bigObjectsImage.convexHull(false);  
BinaryImage filteredImage = convexHullImage.particleFilter(cc);  
  
ParticleAnalysisReport[] reports = filteredImage.getOrderedParticleAnalysisReports();  
for (int i = 0; i < reports.length; i++) {  
    ParticleAnalysisReport r = reports[i];  
    System.out.println("Particle: " + i + ": Center of mass x: " + r.center_mass_x);  
}  
System.out.println(filteredImage.getNumberParticles() + " " +  
Timer.getFPGATimestamp());
```

The particle filter operation uses a series of criteria to determine which detected object should be part of the resultant image. In this case simply the height and width of the rectangles (particles) was used to make sure that only the rectangles were considered. You might want to find a more sophisticated set of filters for your robot program by experimenting with the Vision Assistant on your samples. The criteria is specified using a CriteriaCollection object that has entries for any criteria you wish to apply. It is created here:

```
public void robotInit() {  
    cc = new CriteriaCollection();  
    cc.addCriteria(MeasurementType.IMAQ_MT_BOUNDING_RECT_WIDTH, 30, 400,  
false);  
    cc.addCriteria(MeasurementType.IMAQ_MT_BOUNDING_RECT_HEIGHT, 40, 400,  
false);  
}
```

The program then deletes all the images that were created doing these operations. This is a critical step since the usual garbage collection in Java will not release this memory, since, for performance reasons, is allocated from the underlying C program. Therefore an explicit freeing of the memory is required.

```
filteredImage.free();  
convexHullImage.free();  
bigObjectsImage.free();  
thresholdImage.free();  
image.free();
```