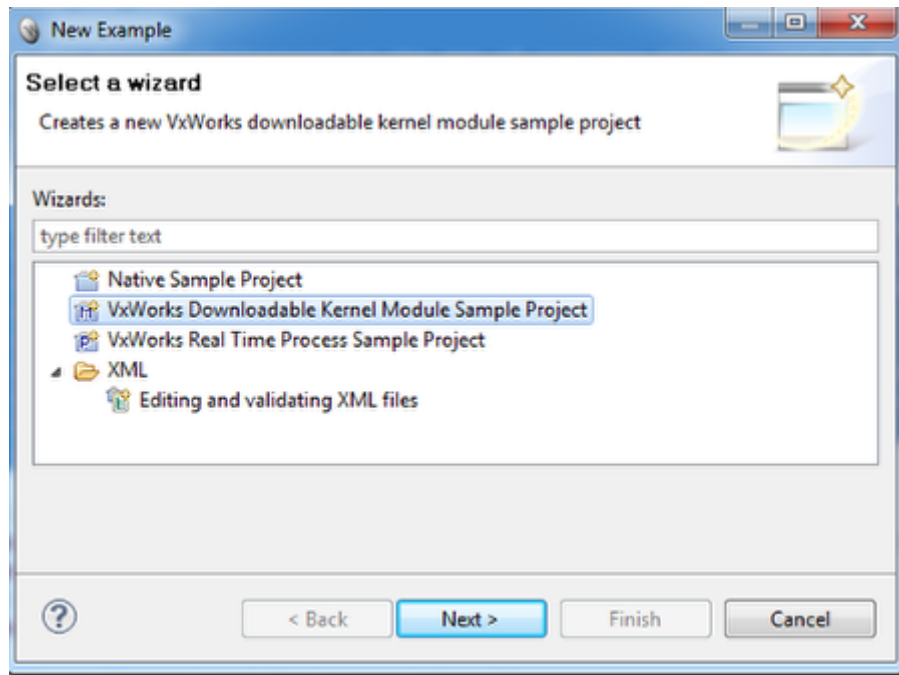


C++/Java Code

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.

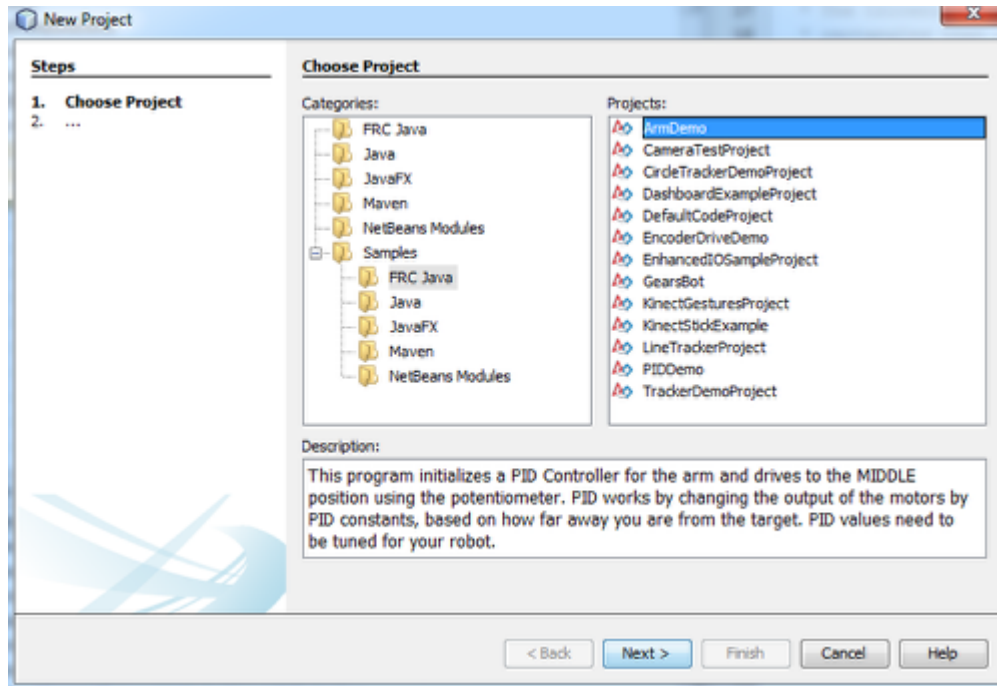
Finding the Example: C++



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.

C++/Java Code

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.

C++/Java Code

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.

C++/Java Code

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()];
```

C++/Java Code

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
}
```

C++/Java Code

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.

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->GetImgImage(), horizontalReport->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_LONG_SIDE, &horizWidth);
        imgMeasureParticle(filteredImage->GetImgImage(), verticalReport->particleIndex, 0, IMAQ_MT_EQUIVALENT_RECT_SHORT_SIDE, &vertWidth);
        imgMeasureParticle(filteredImage->GetImgImage(), 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.

C++/Java Code

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

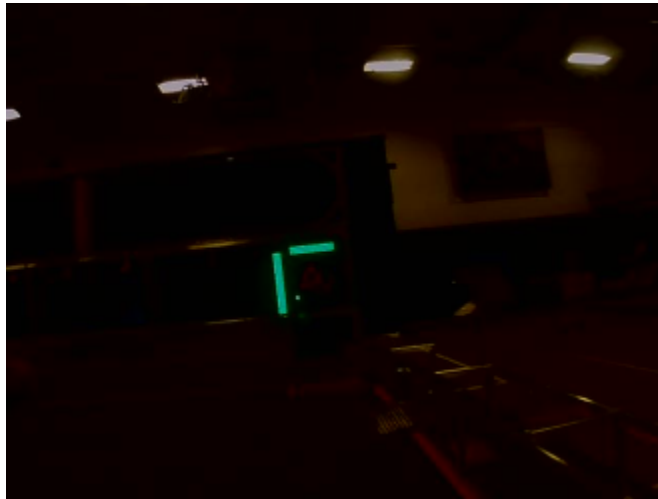
```
// be sure to delete images after using them
delete filteredImage;
delete thresholdImage;
delete image;

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


C++/Java Code

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.