## Guide to the FRC 2010 Sample Source:

This document describes the assumptions that the sample robot code makes about robot construction. It describes how to operate the robot so that the sample code will aim the robot at a target. It describes how the vision code and control code work. Finally, as robots are modified and no longer meet all of these assumptions, this document describes how to troubleshoot and tune the sample code to work as expected.

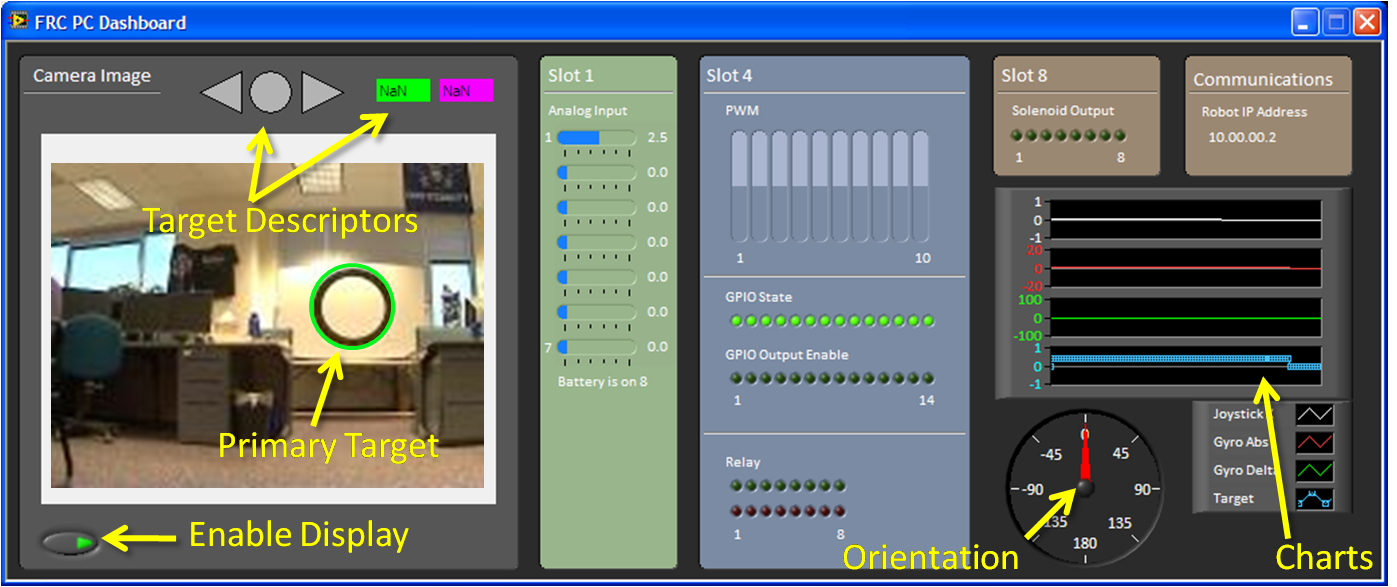
Assumptions about Robot Construction:

The sample code assumes a four-wheel skid-steer robot that has gray wheels mounted on the front axles and white low-friction wheels mounted on the back. It assumes PWM channel 1 on the digital module in slot four controls the left wheels and channel 2 the right. It assumes that the left and right motors are both being controlled using Jaguar motor controllers with the jumper set to coast mode. Note that out of the box, the jumpers are set to brake mode. It assumes that the camera is rigidly mounted on front-center of the robot approximately 24 inches above the floor pointing straight ahead. It assumes that a gyro is mounted on the robot that is connected to analog channel 1 on analog module in slot 1.

This document does not currently describe how to build or mount a target, how to configure or mount the camera, or how to mount the gyro. It also assumes that the development tools have been installed on a computer and the sample source loaded and running. Note that the cRIO doesn’t necessarily have default code deployed and running on it, and if it does, the deployed exe likely differs from the game specific sample source.

Operating the Robot:

The sample source code makes use of an arcade joystick driving system. The first step is to verify the motor channels and motor directions so that the robot drives as expected with the joystick. Pushing forward on the joystick should move the robot forward. Pulling backwards should move it backwards. Pushing left should rotate counterclockwise. Pushing right should rotate clockwise. If these movements are not correct, modify either the source code or wiring to correct the issues. Note that there are two PWM channels, each with a direction input parameter to Drive Open, so there are four total combinations for how the robot will react to the arcade joystick. Until the robot responds correctly to the joystick, it will not be controlled as expected by the vision code.



Next, verify that the image display on the dashboard program shows the images transmitted from the camera. If it does not, check camera power, Ethernet connection, and inspect the Diagnostics page of the driver station to see if any errors are being reported. Also verify that the button beneath the display is in the on position. Verify that the camera is focused properly and gives an unobstructed view of the area in front of the robot. The circle and triangular shapes above the image display give feedback about the target location and quality within the image. When targets are present in the image a green ellipse and zero or more magenta ellipses will be overlaid on the image to indicate the robot has computed the position, angle, size, and quality of the ellipses. The green ellipse has the highest score and the ones marked with magenta have scored high enough to be classified as targets, but lower than the one marked in green. This display will also be useful when setting the minimum target cutoff score to prevent non-target items from scoring high enough to be identified as targets.

When you have mounted a gyro on the robot, verify that the compass-like direction indicator on the dashboard updates correctly as the robot rotates. Pressing buttons 8 and 9 on the joystick simultaneously resets the current orientation to zero and should be used to indicate roughly where the targets are located. The buttons will not work when the driver station mode is autonomous or disabled.

The dashboard also contains a chart with four traces. The top is the arcade Joystick X value passed into Drive. The second is the absolute gyro angle. The third is the delta gyro angle. The fourth displays a dot for each image processed that contains a target. The vertical position of the dot indicates the X offset of the target within the camera image. If an image does not contain the target, or if the target in the image is too blurry, the bottom trace will have gaps for those images. The other traces are useful for measuring overshoot and tuning the robot movement.

At this point you should be able to enable and drive the robot to different locations and determine its ability to identify the targets within the image. Alternately, you can move the targets or camera. If there are locations where the target is not identified, it may be necessary to modify camera or vision parameters as discussed in the tuning and troubleshooting section. If the targeting works reasonably well and with a target outlined in the image, with the robot enabled, press and hold button 3 on the joystick and the targeting information will be used to rotate the robot to aim at the target. If no target is highlighted when the joystick button is pressed, the robot will rotate to point to calibrated angle zero.

How the Vision Code Works:

The vision code responsible for locating targets is based on geometric edge detection. The first step in processing the image is to extract a luminance channel from the color image. This converts bright colors to a light shade of gray and dark colors to a dark shade of gray, etc. The luminance channel is a one byte per pixel grayscale or “black and white” version of the camera image.

The second step is to locate possible ellipses in the grayscale image. There are a number of input parameters such as the one to control the edge threshold – the brightness change that defines an edge in the image and others to filter results based on the size and quality of possible ellipses. Input parameters are covered in more detail in the Tuning section.

To identify ellipses, the IMAQ function searches the image using a grid pattern. When a grid line detects an edge higher than the threshold, if does a local trace following the high contrast edge until it crosses another grid line or connects with another trace. When all grid lines have been searched, the traces are combined and described mathematically as geometric lines, corners, and curves. Using the input parameters, the geometric elements are filtered, scored, and sorted. In this usage, only elliptical shapes will be returned.

After ellipses have been identified, the next step is to normalize the score for size, convert measurements to be based on image ratio instead of pixels, combine concentric ellipses so that they appear as one target with a combined score, and apply a cutoff to rejects circular objects with low scores.

How the Control Code Works:

The vision code measures the location of the target center using the camera and calculates the number of degrees the robot needs to rotate. Using the current gyro position and the angle to the target, it then uses the gyro to rotate to the desired angle. With this approach, it doesn’t matter if the camera images blur during the rotation because the gyro provides the angle instead of the camera. The gyro will measure the angle considerably faster, and with far less lag than the camera, providing a more reliable and robust movement. This control approach will also rotate to point to the angle calibrated as zero if no target is found within the image. If used properly, this can be used to aim in the general direction of the alliance goals even when the camera cannot identify them.

Under the hood, the control code uses a PID algorithm to compute the X joystick value based upon the target center offset. A robot with construction that conforms to the assumptions described above should respond reasonable well with the default PID values. As the robot is modified, changes to the friction with the flooring, center of mass, momentum, and camera mounting will each affect the effectiveness of the control and may require retuning. The motor controller being used will also have a big effect on the response. There are many tuning methods for tuning PID algorithms, but the Tuning and Troubleshooting section at the bottom of this document has a few tips that may also be useful.

The control function also computes an estimated distance to the target using the major radius of the ellipse. If distance is important to the operation of your robot, you may want to verify and possibly calibrate the Proportion based on the size of your target and the location of the camera on your robot. This is also the procedure if you’d like the distance to be estimated in inches, meters, or other units.

## Tuning and Troubleshooting:

Vision:

The dashboard program saves an image once per second to the “Documents\LabVIEW Data” directory. They are named 0.jpg through 59.jpg. If you wish to tune vision processing sample code, it is always helpful to have a good dataset. Since these images are overwritten each time the robot code and dashboard are run, from time to time you may want to copy some of the images to a new location. NI Vision Assistant or another diagnostic tool can then be used to inspect the images for contrast, to extract the luminance channel, and even to detect shapes such as the ellipse. Additionally, Vision Assistant will show the contour lines – the detected edges – to show where a gap may be forming in the edge. Finally, it allows a quick way to check parameter modifications over a large sample of images to verify that the modifications do not have unforeseen effects.

Dim or Bright Lighting:

When there is not enough light, the white areas of the target turn gray. When there is too much light, the black areas of the target turn gray. Both of these conditions reduce the contrast somewhat. To adjust the algorithm to these conditions, lower the Edge Threshold until the contour completely surrounds the target. Another solution, of course, is to turn some lights on or off…

The other way to deal with bright lighting is to correct the camera exposure. There are two good approaches to this. The easy one is to leave Exposure set to Auto and to adjust the Brightness setting on the camera. Lowering Brightness shortens the exposure calculated from the camera’s light meter, and will help to compensate for brightly lit foreground objects or overly dark backgrounds. Increasing Brightness lengthens the exposure and will compensate for backlighting. The other approach is to change from auto exposure to a manually calibrated exposure. The Axis camera has no way to numerically set the exposure, so it will be necessary to correct the composition of the image so that the auto exposure works correctly for the foreground object, and then set the Exposure setting to Hold.



Streaked Lighting:

When bright lights shine on the target such that they can reflect into the camera, they cause glare. Your best approach is probably to correct the lighting artifact. It is difficult and CPU intensive to try and correct a bad image. On a field, the targets will to be mounted vertically, and should not have bright lights mounted behind the robot pointed at the target. This may be more common in shops and classrooms, however.

Blurry Image:

This could be caused by an unfocused camera or by motion. Rotate the camera lens to change focus. Motion blur can be decreased by shortening the time of image exposure. A brightly lit room allows more light into the camera and will shorten the exposure.

Performance of Image Processing Code:

The biggest factor on vision performance is the size of the image. Larger images take more time to decompress and process. Since small images contain fewer pixels and less detail, they will more quickly detect the target up close, but will not work as well at distance. Larger images will work better at distance, but take longer to process. Also note that the small image is grainier, or more pixilated. If high contrast shapes such as squares or rectangles are being incorrectly identified as elliptical, it will often work correctly at higher resolution.

Another performance factor is the edge detection itself. As the Edge Threshold value moves from low to high, fewer edges will be found in the image. Since each edge detected is then fit with a geometric descriptor, classified, and ranked, fewer edges mean faster execution. Setting the Edge Threshold too high will cause an incomplete edge around the target. Setting it too low should still work but will run slower. The LabVIEW implementation of Find Circular Target.vi contains a chart to view the elapsed time to process each image. This is useful for comparing the performance as settings are changed.

It may be relatively obvious, but processing images looking for targets is relatively CPU intensive. By default, the sample code processes images continuously – when in teleop, when disabled, and when rotating. Turning off the image processing loop/task when the target data isn’t needed will free up a very large amount of CPU resources.

Size and Quality Filters:

Several of the input parameters to the shape detector define limits for filtering the identified objects that will be returned. It is sometimes useful to temporarily widen the limits to determine which are responsible for an unreported target. The first filter to consider is the size of the ellipse. The sizes are expressed as a ratio of the image height and by default are set to ignore an ellipse larger than the image height or smaller than 4% of the height. Raising the Min Radius will help ignore false targets, but will limit detection distance.

The second filter to consider is the Min Ellipse Quality Score. The ellipse quality ranges from 0 to 1000 and indicates how well the pixels in the image map to a mathematical ellipse with the given center, radii and rotation angle. Points are deducted from 1000 for each pixel sample that deviates from the elliptical line. The default value of 900 seems to be pretty tolerant, but you may find it useful to lower it further to capture more distorted or badly drawn ellipses. Note that 800 is a relatively low score. The Ellipse score is also size dependent. Small ellipses will score higher than an equivalent ellipse of larger size, so it is necessary to use both the quality and size parameters together and not filter solely on quality.

The third filter to consider is the Min Target Quality Score. The target score is based upon the size, ellipse score, and on whether ellipses are concentric. The target score is less size sensitive and should range from 0 to approximately 100. The default cutoff of 4.0 should detect full-field targets but provide a reasonable filter against badly formed circular shapes within the image. You may choose to be more or less selective based on the environment and distance to target.

Advanced match Options:

It is unlikely that you will need to modify these settings, but if you choose to explore them, here are a few tips. Enabling Scale is equivalent to changing the min and max radius parameters, but you will likely find it more confusing when this is enabled. Rotation can be used to disallow ellipses except in the rotation range you specify. You can also do your own inspection of the ellipses/targets being reported. You probably want to combine the angle and radius information to ensure that slightly elliptical targets aren’t filtered out based solely on rotation. Occlusion can be used to compensate for missing information in the image. The issue with this setting is that you are allowing the algorithm to “fill in” the gaps and then build a score. While this can be used to compensate for obstructed targets, it can easily introduce false targets, so use with caution. Also consider that it gives no measure of where the obstruction is located within the target, and this could indicate a well-defended target. If using occlusion, you will want to limit the amount to something small, ~20% still seems reasonable, and you will likely need to lower the Min Ellipse Quality Score as occluded ellipses score lower in general than complete ones.

Rotation of the Robot using PID Control:

Before modifying values that control the robot, it is a good idea to begin by driving the robot directly with the joystick and observing its response. With a fully charged battery, and the default construction assumptions as described above, the robot rotation will not begin until the X value sent to the WPI Drive function rises to 30% or 40%. You can estimate this by slowly increasing the joystick X value, and inspecting the top trace and gyro trace on the dashboard chart. Changes to the wheels, weight, flooring, battery charge, or motor controllers can have a large impact on this value.

It is also useful, but somewhat more difficult, to estimate the point at which the wheels quickly lose traction and aren’t as effective at rotating the robot. With the default construction, this seems to be around 80%. Knowing this working range for a given robot will help you determine the values you’d like the PID to output. Keep in mind that flooring material, weight distribution, gearing, battery charge, and other factors will affect the control of the robot. If you tune the robot for a given set of conditions, it is wise to record those conditions so that you can restore the values, or at least use them as initial settings for future reference.

Note that there are several standard PID implementations in common use. The LabVIEW PID block uses what is commonly referred to as the academic form of the equation, as shown below. Another common form is referred to as the standard form or parallel form.

Academic PID Formula:



is the gain, and and are the integral and derivative time periods.



Parallel PID Formula:

, , and are the gains.



To convert between these forms, you can see that is equivalent to , to and to .



By following the tuning procedure detailed below you can treat the coefficients as if they are relatively independent, but if you are following your own system, please bear in mind that with the academic form, the proportional gain change will impact other terms, and that the integral coefficient is inverted – meaning that a larger value has less effect and a smaller value more effect.

Tuning the PID coefficients:

The setpoint to the PID is determined by the initial gyro position and the initial angle to the target center. If no target is present when the setpoint is being initialized, the sample code will use 0 degrees as the setpoint, assuming this is calibrated to point to the offensive end of the field. The PID process variable is the gyro angle. The output being controlled is the X value of the joystick and ranges from -1 to 1, however, limiting the range of the output helps to avoid loss of traction and excessive momentum and can also be used to compensate for different battery levels.

Tune the PID using any procedure you are comfortable with. If you have never done this before, below is a simple procedure that should get your robot retuned and aiming reliably in less than five minutes.

Preparation:

Determine if you are going to tune the robot using the target, or the zeroed gyro. If using the camera, drive the robot 15 to 20 feet away from the target and make any necessary camera adjustments to ensure a consistent target is identified. You may find it easier to tune using the gyro zero point – the target isn’t required, and it isolates the control from the vision elements. If using gyro only, aim the robot facing directly towards or directly away from the person tuning the PID. This viewing angle provides for very convenient observations of the robot behavior. In teleop-enabled mode, press buttons 8 & 9 on the joystick simultaneously to zero the gyro. You should see the red line on the dashboard snap to zero at this point. Since the robot will be spinning on the floor, sometimes erratically, be sure to keep it at a safe distance.

It is highly recommended that you use the dashboard or other outputs to view the camera image, gyro values, and joystick X value so that you can better understand the values being sent into and returned from the PID control code.

Simple Tuning Procedure:

General Guidelines

Record all of the initial PID and output limit settings along with the conditions of the tuning. Be relatively consistent with the initial deviations, and if you aren’t sure if an adjustment is an improvement, revert the last change and compare. Take your time, take notes, and test your assumptions as you make changes. For each stage of the tuning, the goal is to adjust a single value up or down observing a particular aspect of the rotation behavior. You generally want to avoid having the robot oscillate and behave erratically, so it is better to choose “safe” values that under react, and adjust those values until they over react, then back off a little. If you are not systematic about changing the values, you can quickly get confused. Retrace your steps until you have a well-behaved rotation, then progress with less aggressive value changes. Also keep your goals in mind. If your PID code can control the robot as well as a good driver, that is saying a lot. And as with all testing, be sure to test your tuned robot with expected changes to the system – different batteries, different carpets, etc.

Proportional Gain:

The proportional Gain is the primary value that determines the robot rotation. It computes an output value that is proportional to the error term. If the difference to the setpoint is zero, no need to steer and it will output a zero output. As the difference increases, the proportional term will increase the output to pivot the robot more quickly in that direction.

Zero out the Integral and Derivative terms to build a simple proportional controller. If you have no idea what value to use for the proportional gain, start with 0.1. Your notes from previous tunings, or values from the sample source are also good initial values. Set the output limit to 0.7 or a previously measured good value. Rotate the robot to point away from the setpoint. If using the camera, place the center of the target at least halfway to the edge of the screen. If using the gyro, rotate it twenty or thirty degrees. You want enough deviation so that the rotation of the robot will reach full speed before ramping down, allowing the PID to exercise the full range of the system. I’ll refer to this as the large error case. We will later fine tune for smaller deviations.

Press button 3 on the joystick to observe how the robot rotates. If the proportional gain coefficient moves the robot in the wrong direction, negate the value. If the gain doesn’t move the robot at all, or moves it too slowly, increase the gain by doubling it. If the robot shoots well past the target, overcorrects, and seems generally out of control, lower it by half. Repeat this procedure, using smaller adjustments if appropriate, tuning the proportional gain until the robot rotates to point at the target or even better, rotates past by a small amount – maybe five to ten degrees. The overshoot will be corrected in a later step. Test both clockwise and counterclockwise response. If there are large differences in the responses in the CW and CCW rotation, try to determine the cause. This can be corrected in SW, but it may indicate an issue with the frame, chain, wheel mounts, etc. If necessary, choose a compromise value that works reasonably well in both directions.

Once you are reasonably happy with the response of the large error case, test with a small error, a smaller angle away from the gyro setpoint. At this point, the small error probably doesn’t rotate the robot at all, and that is acceptable. On the dashboard chart, you should see that pressing the button causes the top trace to jump away from zero, driving the motors by a small amount. Because of friction and nonlinear response of the robot, this joystick value simply isn’t large enough to start the movement. This will be corrected later using the integral term.

First, I recommend modifying the output limit by 0.1 both higher and lower and observing the effect. This acts as an overall limit to the joystick output of the PID, and if you generally feel that the robot should turn more quickly, use a higher number. If you feel that the robot is spinning the wheels too aggressively, try smaller values. It may be useful to once again use the joystick to manually rotate the robot, and compare the top trace response. If you choose a new value, retest and modify the proportional gain if necessary.

Derivative Gain:

The derivative term of the PID is used to reduce overshoot – the amount that the robot rotates past the setpoint. If your proportional setting does not cause overshoot, you can leave the derivative term set to zero. The benefit of using the derivative term in combination with the proportion is that it will allow you to more quickly reach the setpoint. If you do not wish to use the derivative term skip to the integral Gain section.

Use your notes, or the value from the sample code to select an initial value for the derivative coefficient. If you have no other value to use, start with 0.001. Using a similar deviation to what was used in the Proportional tuning, press and hold button 3. If the robot starts to oscillate wildly or starts jumping back and forth before it even reaches the setpoint, release the button and divide the derivative value by two. If the robot still overshoots, double the value. As the derivative term grows, you should see it overshoot less and less. If you start to see a high frequency jump where the rotation stops and restarts, this is another indication that you should lower the derivative value a bit. If it is not possible to avoid overshoot, it may be necessary to lower the proportional gain or lower the output limit a bit, then vary the derivative again. When these two terms plus the output limit are in the correct ranges, the robot will rapidly respond to large deviations and come to rest aiming pretty close to the target. Small deviations will be corrected in the next section.

Integral Gain:

The integral term of the PID is used to correct small errors. When the robot is only off target by a few degrees, the proportional term will contribute a small joystick movement, but it often is not enough to overcome friction. The integral term will slowly increase the joystick value over time until the robot overcomes friction, bumps or rough spots in the floor, etc.

Use your notes or the value from the sample code to select an initial value. If you have no other value to use, start with 0.01. Using a small deviation, press and hold button 3. If the robot swings wildly, release the button and decrease the effect of the integral term. Note that since LabVIEW uses the academic form, you actually want to increase the time value to decrease the effect. If the robot takes a very long time to start moving, increase the effect. In LabVIEW, this would mean lowering the time value. It may be useful to press the button and pay attention to the top trace on the dashboard chart. With zero for an integral term, the small error would produce a small joystick value that would never change. As you increase the effect of the integral term, it will increase the rate at which the joystick increases. When the effect gets too high, it moves the joystick too quickly, maxing out the joystick, overshooting, then oscillating. When properly tuned, given the torque and weight of the typical robot, you should see small errors corrected in approximately 0.5 seconds.

Testing:

At this point it would be wise to test thoroughly over the conditions you expect the robot to encounter. If the robot will collect objects and gain weight, or drop objects and shed weight, test those conditions. Test different distances and different surfaces to learn how the current tuning behaves. Take good notes, and if you need to retune, compare to determine which coefficients needed to be modified. More than likely, you can find a single set of coefficients that work pretty well for most conditions. If using the LabVIEW front panel to modify the coefficients, don’t forget to right-click and Make Current Values the Default and save changes. If using C++ or Java, similarly make sure that your robot will actually use the values you’ve discovered.