
KiwiLight Manual

By Brach Knutson

V. 1.4

Table of Contents

Overview

KiwiLight is a smart vision solution designed for the FIRST Robotics Competition by FRC Team 3695: Foximus Prime. It is a Linux application designed and developed for use on the Raspberry Pi and uses anything from simple USB webcams to advanced computer vision cameras to solve FRC vision challenges at high framerates. This manual covers the basic algorithm that KiwiLight uses to achieve its functionality and how to use KiwiLight on an FRC robot.

To learn how to fully implement computer vision functionality onto your robot using KiwiLight, start at the “KiwiLight on the Raspberry Pi” chapter and read until the end of this manual. These chapters will discuss how to configure the components on the robot and then the robot code required to drive them.

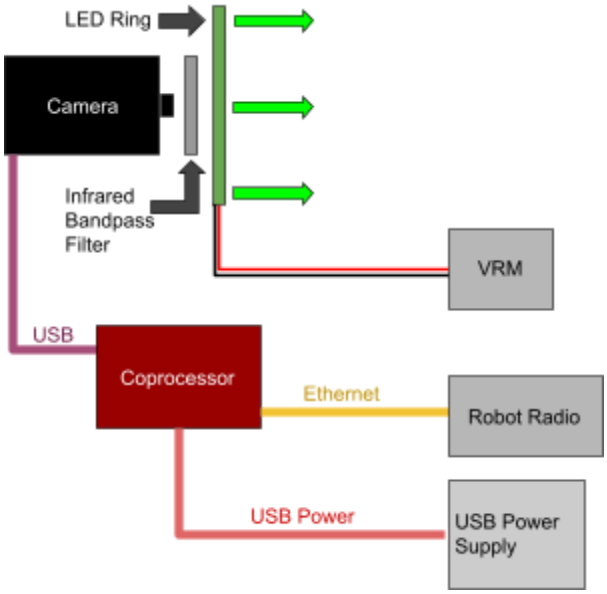
KiwiLight is NOT the same thing as Limelight. Those who are looking for a Limelight camera can find one at limelightvision.io.

KiwiLight on the Raspberry Pi

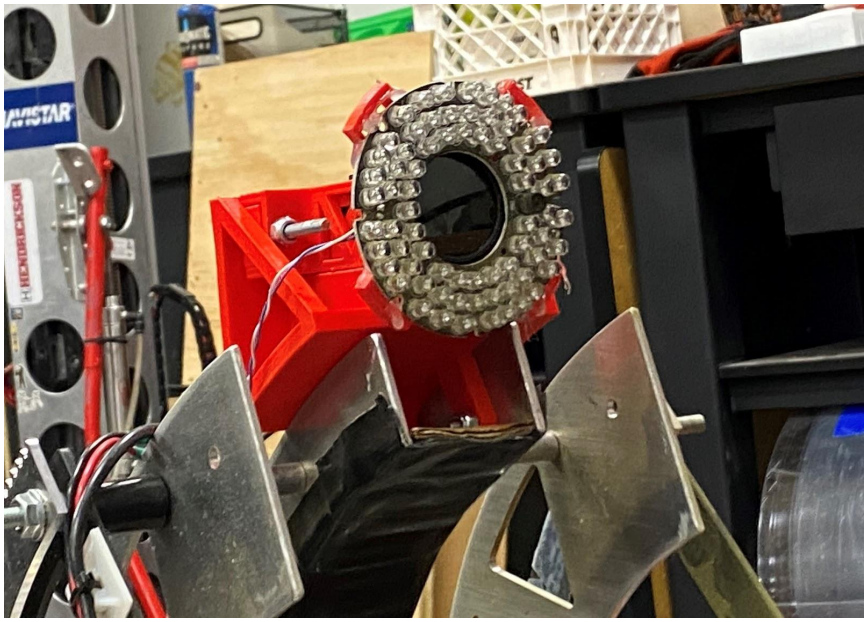
This chapter covers the KiwiLight app. It will discuss how to download and install it, how to use it, and features it offers.

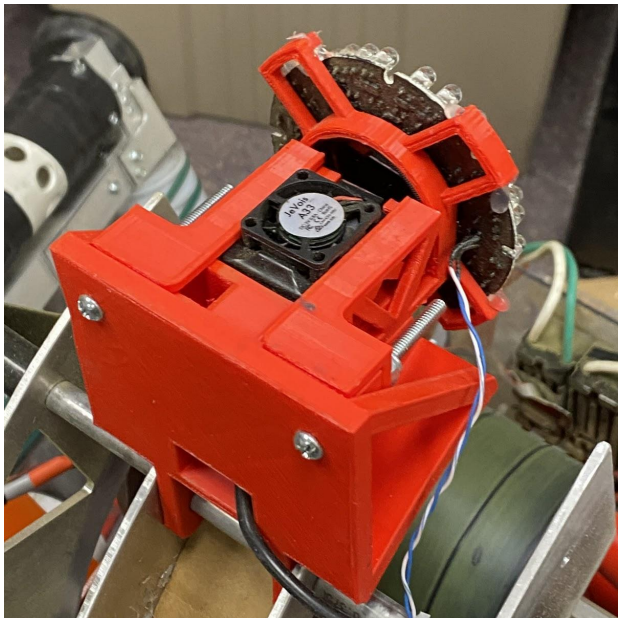
Setup on the Robot

In order for KiwiLight to work on your robot, it requires a few things to be set up and configured correctly. KiwiLight uses an advanced take of the standard FRC vision algorithm described in the previous chapter, meaning that it looks for direct light sources to find targets. That being said, the camera requires an LED ring in front of it to shine light at the retro-reflective targets. This LED ring can be any color, but green or infrared is recommended. The camera should be connected to a coprocessor that will run KiwiLight. A Raspberry Pi is recommended because it is cheap, small, and lightweight, but has enough power to easily do vision calculations. To power the Pi, you can use a portable phone charger with a USB output. Powering the Pi with the robot's VRM is not recommended because the hefty and inconsistent power draw of the robot can cause power fluctuations in even the VRM (especially during brownout), which could make the Pi unstable. This could cause programs running on it to freeze or crash. The Pi should also be connected to the robot router via Ethernet so that it can communicate with the RoboRIO. Below is a diagram of a functional vision setup. Note that the infrared bandpass filter in the diagram should only be included if an infrared LED ring is being used.



A diagram of the vision setup on the robot. The Infrared Bandpass Filter should only be included if an infrared LED ring is being used.





Shown Above: Sample vision setup. Infrared ring and bandpass filter(upper left), Jevois A33 camera in mount (upper right), Raspberry Pi Coprocessor (lower center). Orange cable is USB power cable, Yellow cable is an Ethernet cable, and the black USB cable is the camera cable.

Installing the App

To install KiwiLight on your Raspberry Pi, follow these steps:

- *Download the latest version of KiwiLight*

  Releases of KiwiLight can be found at github.com/wh1ter0se/KiwiLight/releases/latest. 5 downloads are available for each release. Download KiwiLight-x.y.z-RaspberryPi.zip if you are using a Raspberry Pi, or "Source Code (zip)" if you are using something else.

- *Run the Installer*

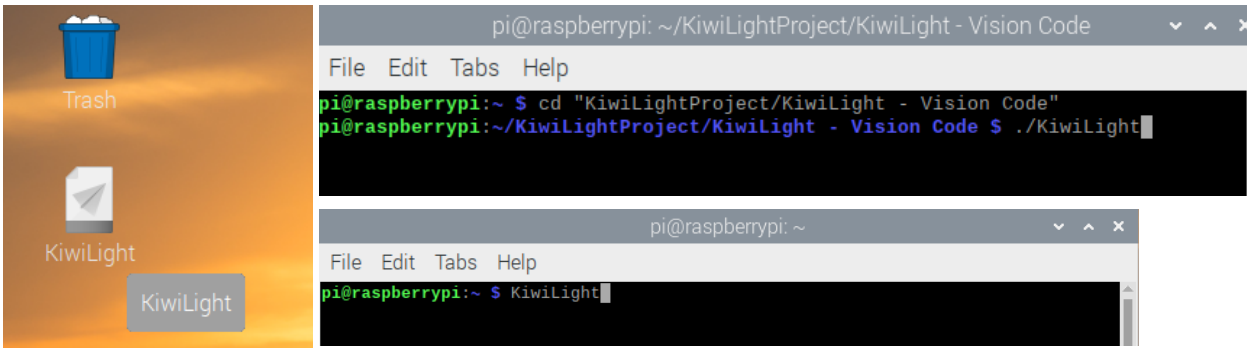
  Extract the .zip file that you just downloaded. Then, in the terminal or command line, `cd` into the unzipped directory with the command:
  `cd <extracted directory>`

  If you downloaded the "Source Code (zip)" file, `cd` further into the `"KiwiLight - Vision Code/"` directory, which should be directly inside of the unzipped directory. Because there are spaces between the words and dash, you will need to put quotation marks around the directory:
  `cd "KiwiLight - Vision Code"`

  The directory should contain a shell script named "install.sh". Run that using the command `sh install.sh`. This script may take a while to run, especially if using the "Source Code (zip)" package.

  After running that script, KiwiLight is installed on your coprocessor.

Running KiwiLight



Shown above: KiwiLight desktop icon (left), KiwiLight source launch command (right top, ONLY if running KiwiLight from source), KiwiLight command (right bottom)

There are multiple ways to run KiwiLight:

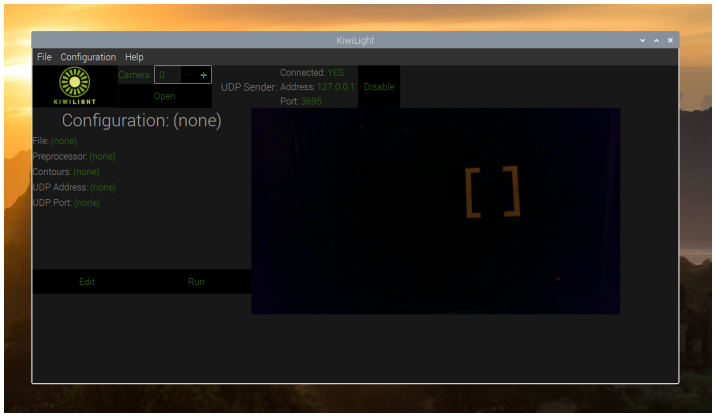
- *If you downloaded Raspbian.zip or Ubuntu.zip:*
  Double-click on the "KiwiLight" desktop icon, if you let the installer generate one, to launch KiwiLight. If the system asks you what you want to do with the file, click "Execute." If no desktop icon is available, enter the command `KiwiLight` in the terminal.

- *If you downloaded "Source Code (zip)"*
  In the terminal, `cd` to the KiwiLight project folder (where your install.sh file was). Then, enter the command `./KiwiLight`.

  If the file "KiwiLight" does not exist, then KiwiLight needs to be compiled. To compile KiwiLight, run the following commands in order:

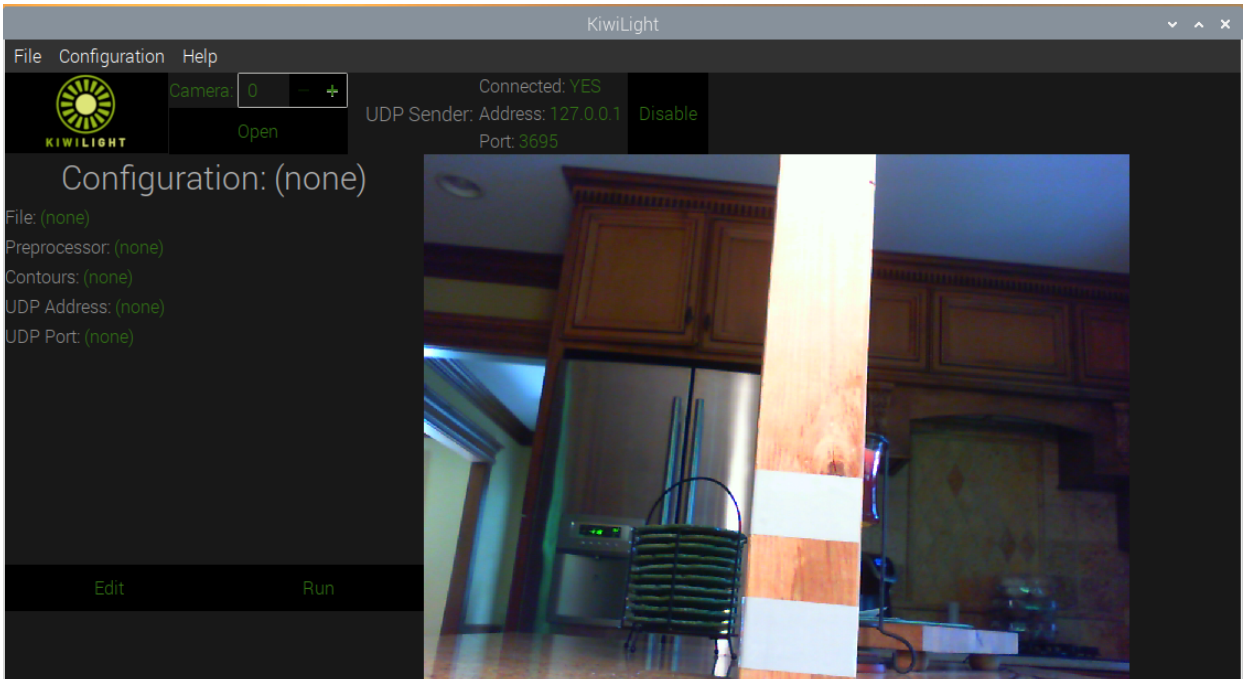
  - `make setup`
  - `make -j4 KiwiLight`

After launching KiwiLight, you should have a window that looks like this:

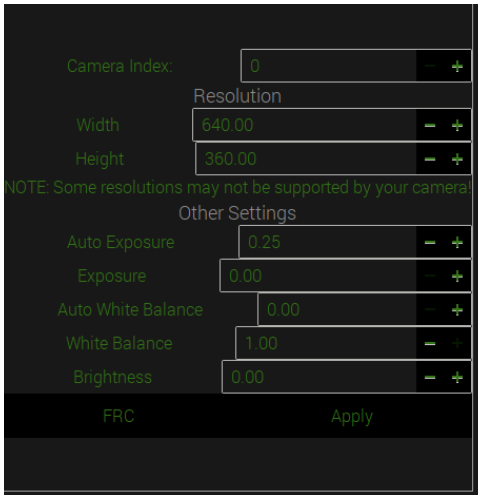


Using KiwiLight to automatically learn targets

One of KiwiLight's main features is its ability to learn targets in a matter of seconds. It does this by taking multiple images and analyzing each contour for the parameters that identify it as a part of a target. This section will guide you through how to use that feature.

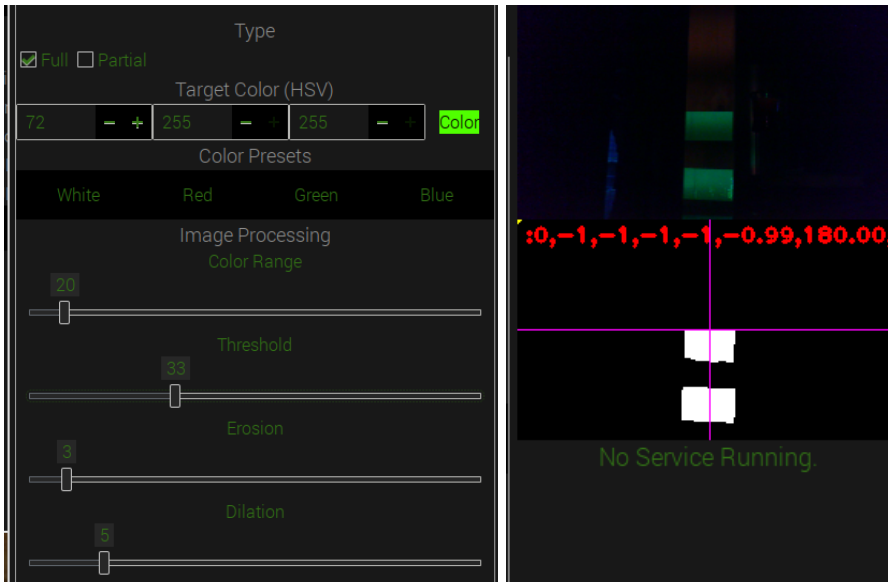


First, load up a new configuration by clicking "File" then "New Configuration" and go to the Camera tab. Set the resolution of the camera to whatever you would like. Lower resolution is usually better. Then, apply FRC settings by pressing the FRC button, then the Apply button.

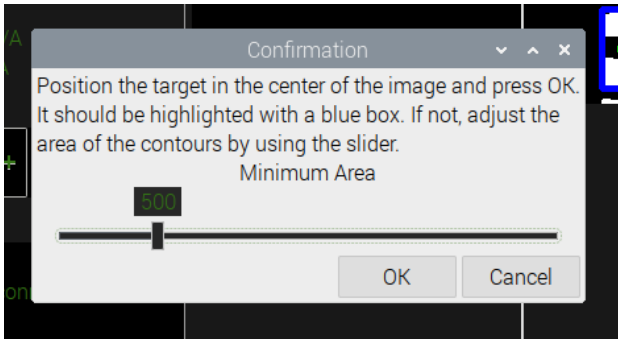


Then, go to the "Preprocessor" tab. If you are using a smart camera, such as a JeVois A-33, that is configured to preprocess the image for you, switch it to "partial" mode. Otherwise, set the target

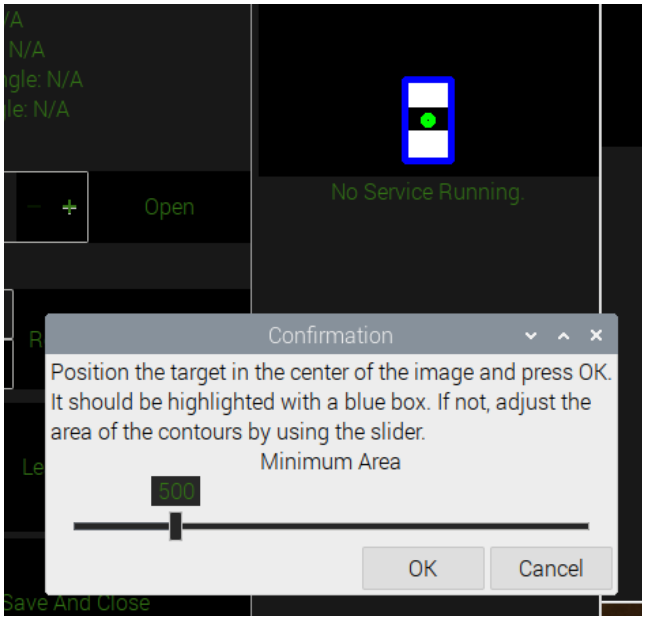
color (in this example, we used the green preset) and gradually lower the “Threshold” slider until the target chunks are clearly visible in the output image (the bottom one).



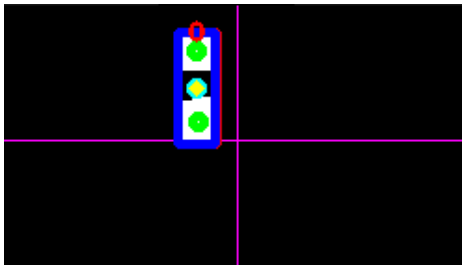
After that, go back to the Overview tab. Press the “Learn Target” button. The below confirmation dialog should pop up:



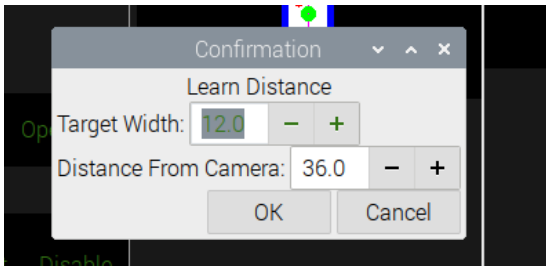
Adjust the “Minimum Area” slider until the target is enveloped by a blue box:



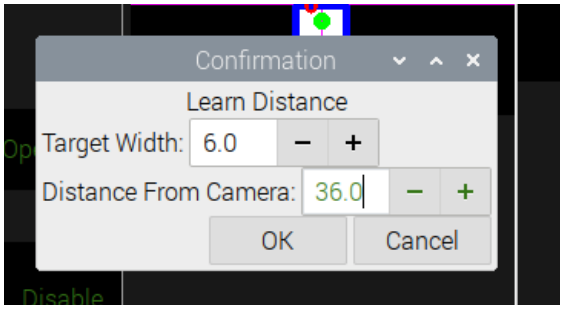
Then, press OK. KiwiLight will begin working on learning the target. After a few seconds, the process should finish and KiwiLight should recognize your target as a valid target.



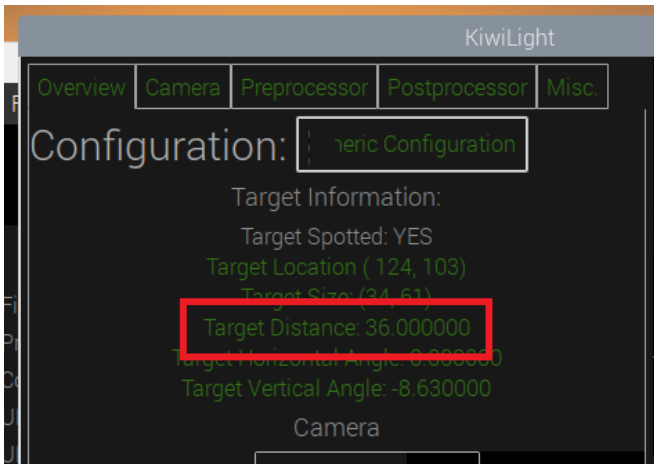
In some cases, you may wish for KiwiLight to calculate an approximate distance from the camera to the target. This is useful in cases where KiwiLight is aiming a turret that shoots a projectile that is affected by gravity (like in the 2021 FRC game). Note that automatic distance tuning will only work once the target has been learned. To easily tune KiwiLight's distance calculation, click the "Learn Distance" button in the Overview tab. A dialog like the one below should appear.



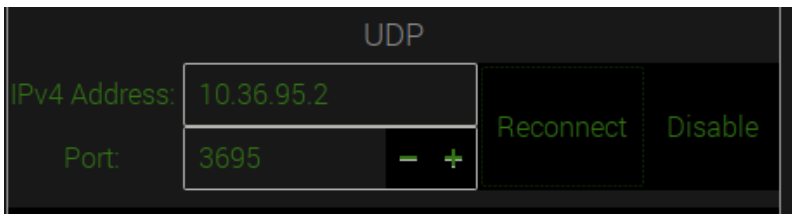
Fill out the “Target Width” and “Distance From Camera” fields. These can be in whatever unit you would like, but they must both be in the same unit. The distance that KiwiLight reports will then be in the same unit. Below, the fields are being filled with measurements in inches.



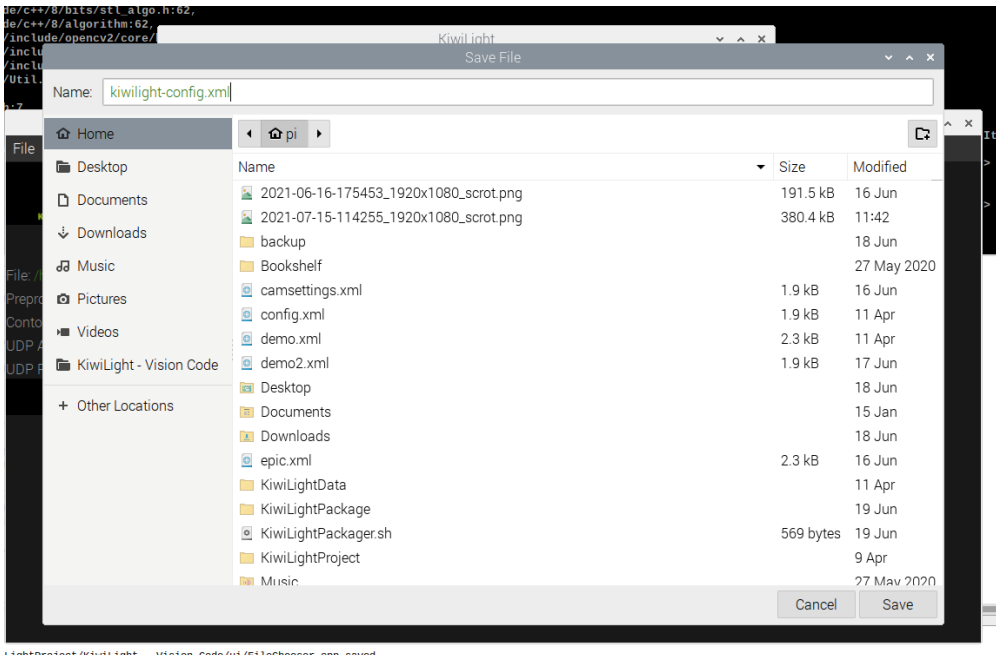
Then click OK. After a few seconds, KiwiLight’s learning process will finish. Check the distance reported by KiwiLight in the “Target Information” section in the Overview tab. It should read the distance that you set in the “Distance From Camera” field



After that, KiwiLight must be configured to send the target data to the RoboRIO so that it can be used by the robot code. In the Overview tab, enter the RoboRIO’s IP address (which can be found on the Driver Station UI), and a port number that is not blocked by the Field Management System. To see ports blocked by FMS, consult the current season’s game manual. Take note of the port number that you are using because you will need to initialize KiwiLight’s `SubsystemReceiver` object in your code with the same port number.



After that, save and close your configuration by clicking the “Save and Close” button. Use the file dialog that pops up to select where to save your new configuration file. Then, click “Save.”



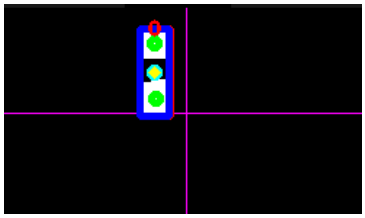
Now, your coprocessor is configured and ready to do vision for your robot. To make your vision system fully functional, see the “KiwiLight on the Robot” chapter, which will guide you through how to code your robot to receive target data from the coprocessor using KiwiLight’s `SubsystemReceiver` class. For more reading about the KiwiLight editor, see the below section, “Working with KiwiLight Configurations.” To learn a little more about how to use your configuration, such as how to start it automatically when the coprocessor boots up, see the “Using Configurations” section.

Working with KiwiLight Configurations

This section will guide you through all of the features that KiwiLight has to offer.

- *To create a new vision configuration:*
  To create a new configuration, either click the “Edit” button when no configuration is loaded, or click “File” then “New Configuration”.

- *Running Vision Configurations:*
  To open a configuration, click “File”, then “Open Configuration”. A file dialog should appear. Use that file dialog to select the configuration you wish to load. KiwiLight will load the selected configuration and run it. An output image like the one pictured below will be provided on the screen.



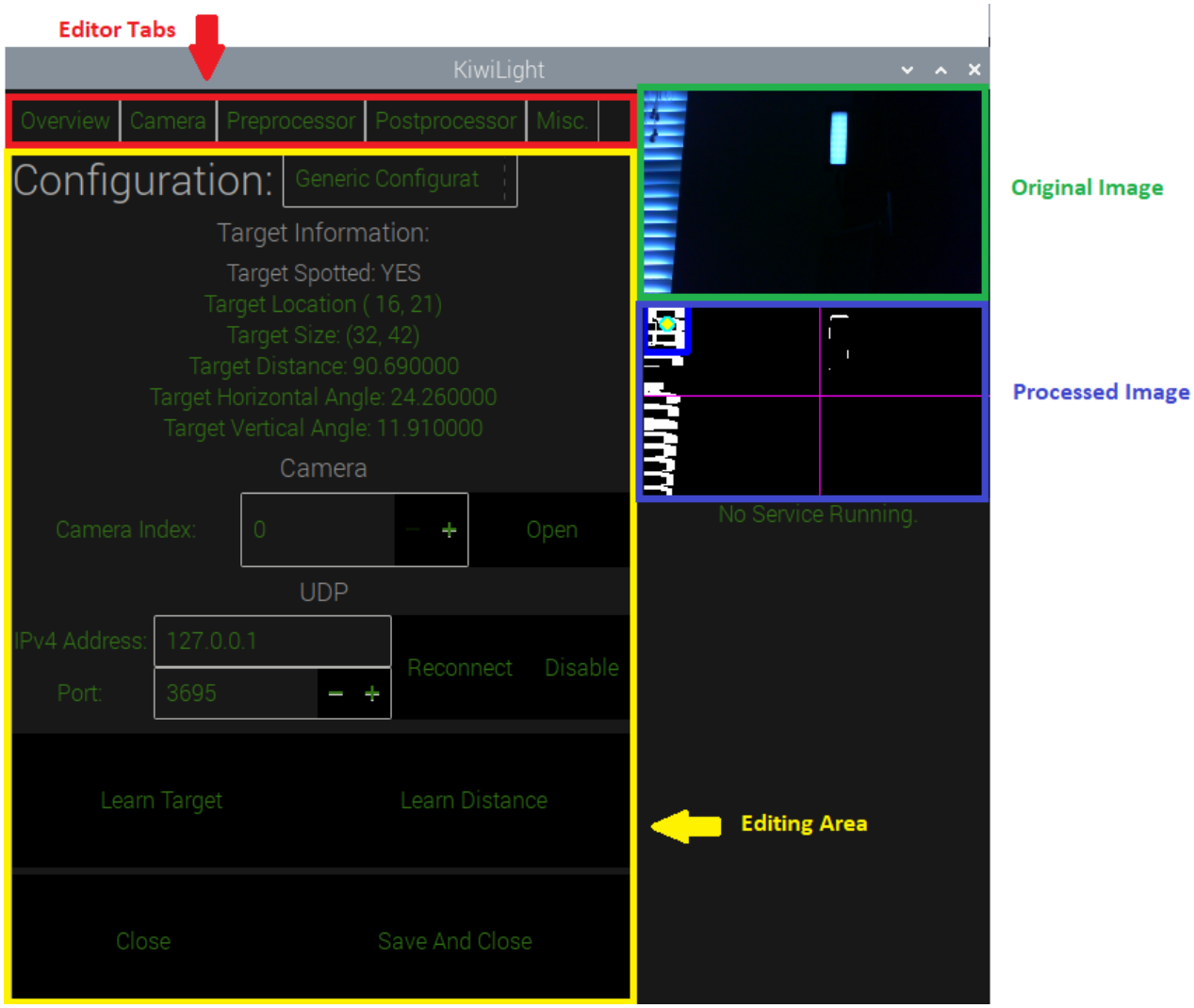
This output image provides important information about the data that KiwiLight is sending to your robot. The white chunks are the parts of the image whose color matches the target's color. In addition, each markup in the image means something.

- The blue box around the target marks a valid target. The blue box also has a light blue dot in the center of it.
- The yellow dot inside of the light blue dot marks the target as the one that KiwiLight is "focusing" on, and therefore the one that KiwiLight is sending data for. Targets with only a light blue dot are ignored. KiwiLight focuses on the targets that are closest to the robot's center, marked by the pink crosshair.
- The green dots mark contours that are valid enough to be considered part of a target.
- The pink crosshair marks the perceived robot center. Any angle calculations will be done with respect to those lines. They are also used to determine what target KiwiLight should focus on.

- *Editing Vision Configurations:*

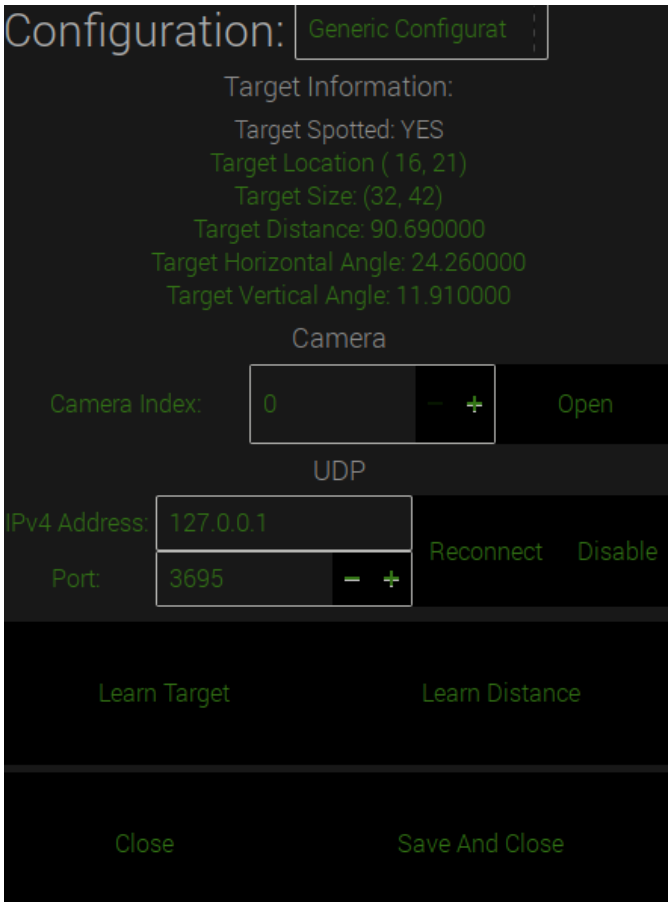
  To edit a configuration, first open it. Then, click the "Edit" button to open the editor.

  The editor will appear as another window. The main part of the window features multiple editors that can be selected using the different tabs at the top. These sub-editors provide settings of a specific part of KiwiLight's algorithm that can be changed to configure or fine tune your vision configuration.



The layout of the KiwiLight configuration editor.

- **The Overview tab**

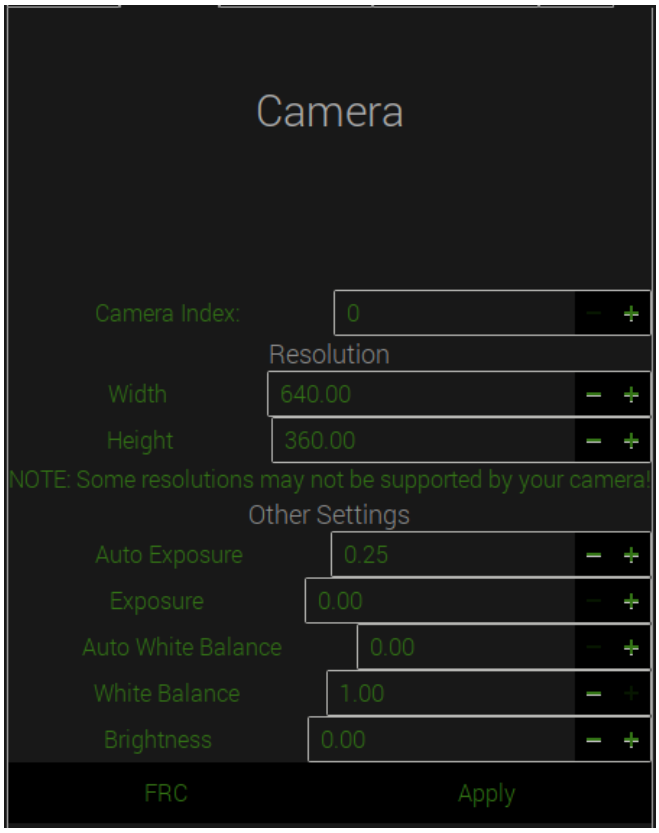


This tab contains important settings as well as the buttons that allow the user to learn a target and close the editor. The table below outlines a few settings that this editor allows the user to modify.

Feature	Function
The “Configuration Name” text box	This text box is where the user can set the name of the configuration. This name will be displayed on the KiwiLight main window whenever the configuration is loaded and will also be displayed in the terminal whenever the configuration is run headlessly.
The “Camera Index” text box	This text box is where the user can specify the camera that should be used to search for the target. Users should only need to change this if more than 1 camera is connected to the coprocessor to specify which camera the configuration should use.
The “IP Address” and “Port” fields	These fields allow a user to set the IP Address that the target data will be sent to, and the port to use to send that data. The port set in this field should match the port that the `SubsystemReceiver` object is initialized with in

	the robot code. The `SubsystemReceiver` class will be covered in the KiwiLight on the Robot chapter. To set recently changed IP address and port settings, the user should press the “Apply” button which is located next to the two fields.
The “Learn Target” button	The user should press this button to initiate the target learning process when the preprocessor is configured using the settings in the Preprocessor tab. More about learning targets is provided later in this chapter.
The “Learn Distance” button	The user can press this button to initiate the target distance learning process after the target has been configured. More about learning distances is provided later in this chapter.

- **The Camera tab**

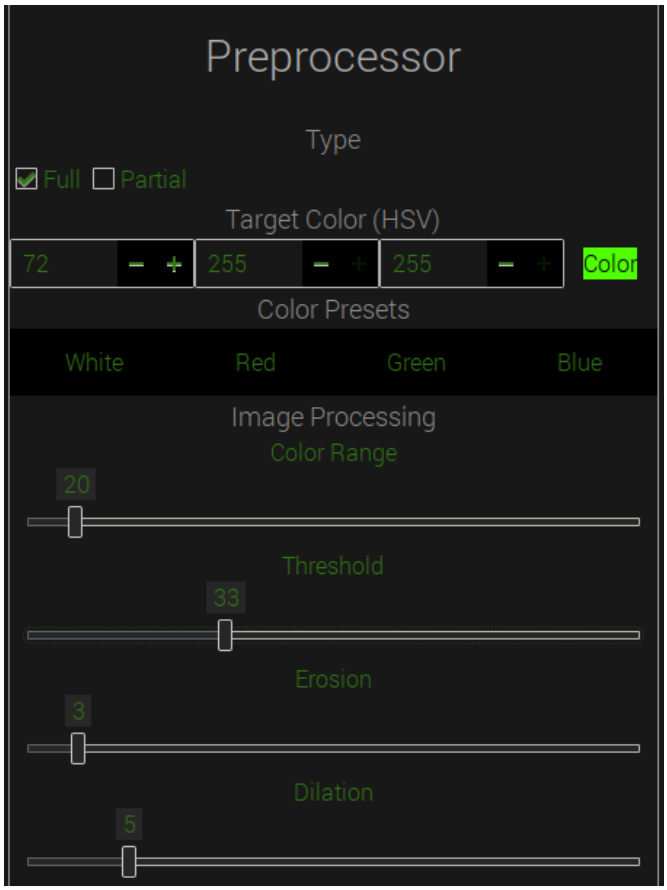


The camera settings tab allows the user to edit the way that the camera takes images. KiwiLight uses OpenCV to set these camera settings, so there is no guarantee that all of the settings will work. The table below outlines the settings that the user can modify in this sub-editor.

Feature	Function
Camera Index	This field is where the user can specify the camera that should be used to search for the target. Users should only need to change this if more than 1 camera is connected to the coprocessor to specify which camera the configuration should use.
Width	This field can be used to set the width component of the camera's resolution.
Height	This field can be used to set the height component of the camera's resolution.
Auto Exposure	This field can be used to set the automatic exposure setting of the camera. For FRC applications, it is recommended that auto exposure be disabled.
Exposure	This field can be used to set the absolute exposure setting of the camera. For FRC purposes, this value be set to the lowest possible value where direct light sources can still be seen.

Auto White Balance	This field can be used to set the automatic white balance setting of the camera. For FRC settings, it is recommended that auto white balance be disabled.
White Balance	This field can be used to set the white balance setting of the camera. In FRC, this can be any value as long as the auto white balance feature is disabled.
Brightness	This field can be used to set the brightness of the camera. For FRC, this value should be set low, but not so low that direct light sources can no longer be seen. A value set too high can cause the image to be noisy.
The “FRC” Button	This button sets recommended FRC settings that will work for most generic webcams, such as the Microsoft LifeCam HD 3000. This button does not apply the settings; you will need to hit the “Apply” button at the bottom of the editor.
The “Apply” Button	This button applies recently set camera settings.

- **The Preprocessor tab**

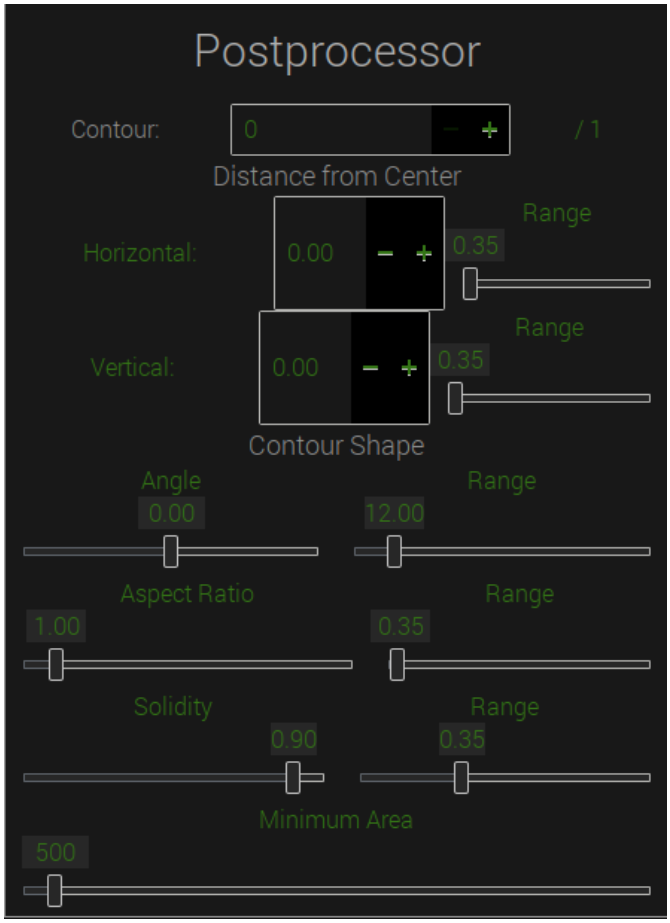


The preprocessor works with the raw images, filtering them down to the properly colored chunks. In this editor, the user can change the color of the target and adjust some other important settings. The table below outlines some settings that the user can modify in this sub-editor.

Feature	Function
The “Type” selector	This selector allows the user to specify the mode that the preprocessor will run in. Two modes are possible: - **Full Mode:** This mode should be selected if a generic webcam such as the Microsoft LifeCam HD 3000 is being used. The preprocessor will perform all image manipulations and calculations. - **Partial Mode:** This mode should be used if a smart machine vision camera such as a JeVois A-33 is being used. The camera must be configured to automatically process the image into a binary one (all pixels of the correct color are white, everything else is black). In partial mode, KiwiLight will skip all image manipulations and calculations with the exception of resizing the image.
The target color selector	This allows the user to select the color that the target will appear as. This should be the color of the light that shines on the target.

	This color is entered in the HSV colorspace; the leftmost field is the hue of the color, the middle field is the saturation of the color, and the rightmost field is the value (a.k.a brightness) of the color. In addition to the 3 fields, 4 preset color buttons are provided.
The color range slider	Because no color will be the exact color that is specified in the fields described above, this slider allows the user to choose the amount of error that KiwiLight will allow when filtering out improperly-colored pixels.
The threshold slider	This slider allows the user to specify how much threshold should be applied to an image before it is analyzed. The greater this value is, the less noise will be present in the fully processed image, but the target will need to be brighter to be seen..
The erosion slider	This slider allows the user to specify how much to erode the chunks in an image before analyzing it. The more an image is eroded, the less noise that will be present, but big objects like the target may be wispy and hard to discern.
The dilation slider	This slider exists to solve the wispiness problem that erosion creates. After eroding the objects in an image, they are dilated to make them less wispy and more perfect. It is recommended that this value be your erosion value plus 2 or 3.

- **The Postprocessor tab**

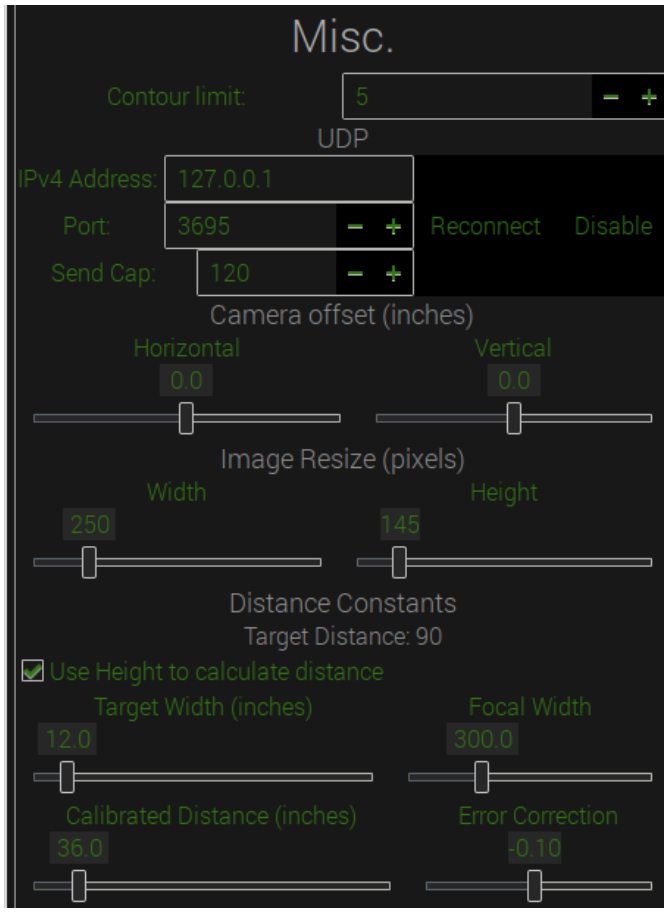


The postprocessor tab allows the user to fine-tune aspects of the target itself. This tab should only be used for fine tuning. Manually doing target recognition with this sub-editor can get very difficult. Target recognition should instead be done with the "Learn Target" button in the overview tab.

Feature	Function
The "Contour" field	This field is used to select the contour that is being edited. Order of selection goes from the top-leftmost contour to the bottom-rightmost contour.
The "Distance From Center" fields	These 4 fields allow the user to configure where the selected contour should be located relative to the center of the target. All of these values are measured in terms of the target's width in image space (i.e a "Horizontal" value of 0.3 means that the contour is 0.3 target widths to the right of the center of the target). When tuning these, remember to account for target imperfections. For example, viewing the target at an angle from the left side may cause the target to appear thinner than it actually is.

	The “Horizontal” field configures the horizontal distance to the center of the target. The “Vertical” field configures the vertical distance to the center of the target. The “Range” fields configure the allowable error of their respective fields. Contours will pass property tests if the property being tested is within the value plus or minus the range.
The “Angle” fields	The angle slider and its respective range slider configure the angle of tilt of the selected contour. Its “Range” slider configures an allowable error for this value. These values are measured in degrees.
The “Aspect Ratio” fields	The aspect ratio slider and its respective range slider configure the aspect ratio (*width* / *height*) of the selected contour.
The “Solidity” fields	The solidity slider and its respective range slider configure the solidity of the selected contour. Solidity is the ratio of a contour’s actual area to its bounding box’s area. For more, see the Algorithm chapter.
The “Minimum Area” slider	The minimum area slider configures the minimum area that a contour has to have to be considered a part of a target. This value helps to filter out noise that may be present in the image.

- **The Misc. tab**



The miscellaneous settings tab is where other miscellaneous settings are configured. These settings do not belong in any of the other categories, but are still important to the functionality of the configuration. The table below outlines settings that the user can modify in this sub-editor.

Feature	Function
The “Contour Limit” field	This field allows the user to specify the maximum number of contours that may be considered for targethood. If the number of contours in the image exceeds this number, then only the biggest contours are analyzed. This setting is extremely important to optimizing the functionality of the configuration. Noisy images can potentially cause numerous false contours depending on how forgiving the postprocessor is, which can slow down the configuration greatly. This setting strides to limit that effect.
The “IP Address”, “Port” and “Send Cap” fields	These fields, with the exception of the “Send Cap” field, are exact clones of the ones in the “Overview” tab. Similar to the Overview tab,

	a “Reconnect” button is available to apply the settings that the user has entered. A “Disable” button is also present so that the user can disable data transfer temporarily. Finally, the “Send Cap” field is present. This field allows the user to configure the maximum number of data packets that can be sent per second. This helps the user manage the traffic that is going through their router. Especially on the FRC Robot Radios, data packet overloads can cause robot communication problems which can compromise human control over the robot. This setting seeks to limit that effect.
The “Camera Offset” fields	These fields allow the user to configure the offset of the camera from the robot’s center. This allows KiwiLight to calculate horizontal and vertical angles as if it was looking at the target from the center of the robot, making it easier for programmers to code functions that align the robot to a target of some sort.
The “Image Resize” fields	These fields allow the user to configure the size that the image is converted to before it is analyzed. By making the image smaller, KiwiLight is able to run faster. However, making the image too small can eliminate potentially important image details.
The “Distance Constants” fields	These fields allow the user to configure the values that KiwiLight uses to calculate the distance from the camera to the target that it is focusing on. Perhaps the most useful of these fields is the “Error Correction” slider which can be used to compensate for error in the calculation. If at any time you notice that the distance is being reported incorrectly, that slider should be adjusted until the distance reads the correct value.

Using Configurations

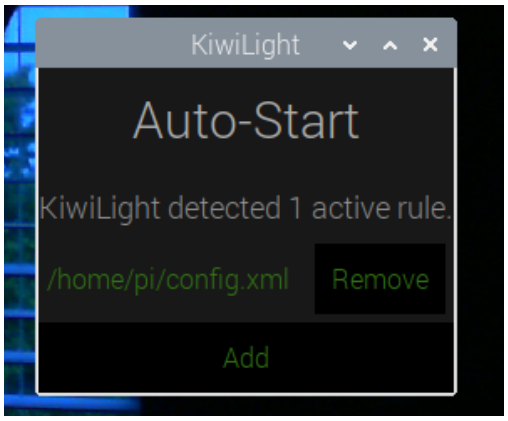
Running a KiwiLight vision configuration is easy: simply open it by clicking on “File”, then “Open Configuration”. Then, use the file dialog that pops up to select the configuration that you wish to load. Automatically, the configuration will run and begin sending data to the RoboRIO. However, there are a few other ways to run your configuration:

- *Running your configuration headlessly (without the GUI)*
  Running a configuration headlessly means running it without the GUI attached. This can dramatically increase the speed at which the configuration runs. You can run a configuration headlessly one of two ways:

  First, you can simply press the “Run” button, which is located next to the “Edit” button. If a configuration is loaded, it will be run headlessly. Otherwise, KiwiLight will prompt the user to select a configuration to open.

  Second, you can launch the configuration using the terminal with the command:
  `KiwiLight -c <configuration file path>`
  If you built KiwiLight from source (you downloaded “Source Code (zip)”), then you will need to execute this command inside of the KiwiLight project directory (where install.sh is) and put a `./` in front of the command:
  `./KiwiLight -c <configuration file path>`

- *Running your configuration automatically when the coprocessor boots up*

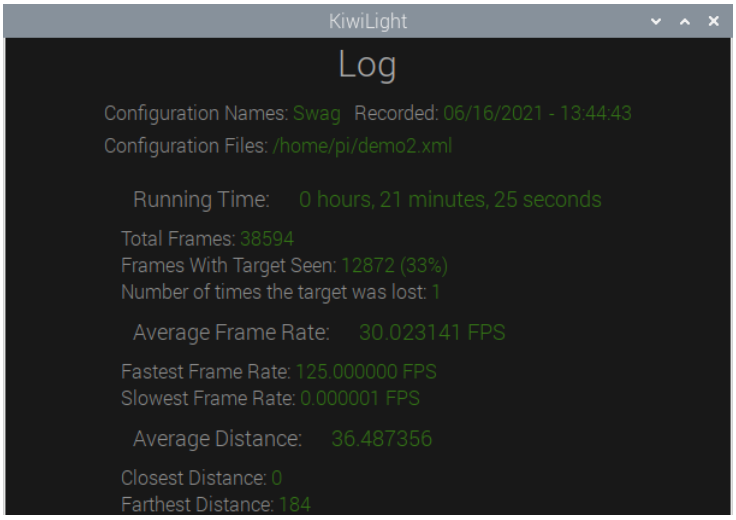


Especially at an FRC competition, it can be tedious and inconvenient to connect to the coprocessor to manually start your vision configuration before every match. That is why KiwiLight offers the option to automatically start a configuration when the coprocessor boots up. To do that, follow these steps:

In the KiwiLight GUI, click “Configuration”, then “Configure Auto-Start”. A window like the one pictured should pop up. Click the “Add” button, then use the file dialog that pops up to select the configuration that you want to have started automatically.

After doing this, your configuration(s) should automatically start when your coprocessor boots up. Note that if you are having multiple configurations start automatically, they must each use a different camera in order to work.

- *See how KiwiLight is performing*



Every time a configuration is run headlessly, it generates a KiwiLight log file. These files are stored in the “KiwiLightData/logs” directory, which can be found from your HOME directory (/home/<user>). To view these files, follow these steps:

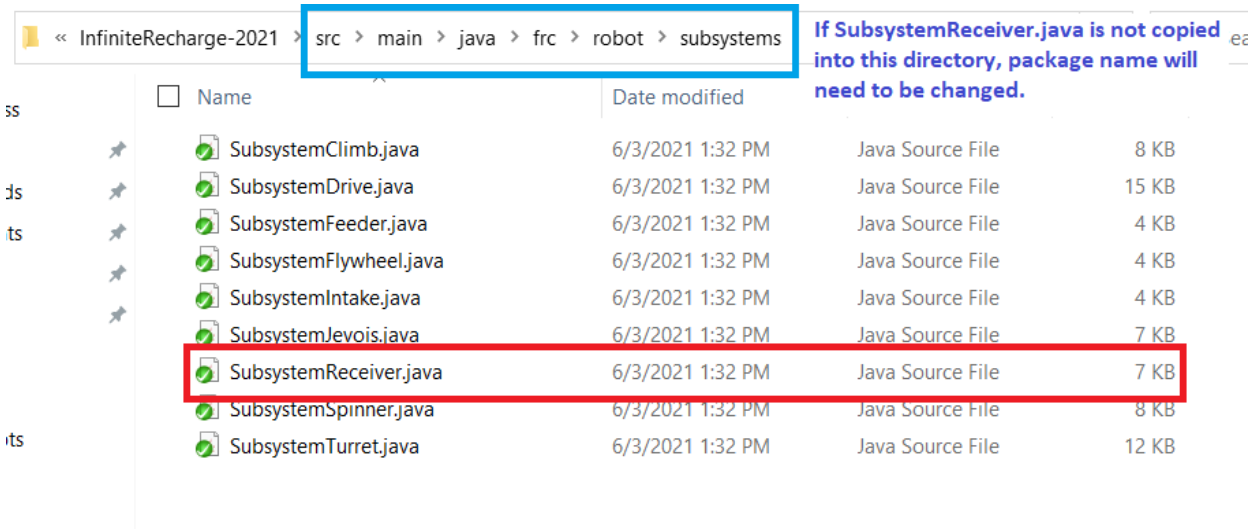
In the KiwiLight GUI, click “Configuration”, then “View Log”. This should open a file dialog. Use it to select the log file that you wish to view. KiwiLight will then create a window that displays the information stored in the log, which looks something like the window pictured.

KiwiLight on the Robot

KiwiLight does all of its heavy calculations on a coprocessor. After those are done, the results need to be received by the robot. This work is done by the `SubsystemReceiver` class. This chapter will cover that class, how to implement it into your robot code, and how to use it to make your robot align to targets.

How to add KiwiLight to your Robot Project

KiwiLight is easy to add to your robot. To start, download the SubsystemReceiver.java file from the latest release of KiwiLight. *If the release is older than 1.4,* download the KiwiLight-X-RobotCode.zip file and use the file system to search for SubsystemReceiver. Copy and paste SubsystemReceiver.java into your robot project, preferably into the "subsystems" folder. If you did not put the file into the "subsystems" folder, you will need to correct the `package` name at the top of the file.



Putting SubsystemReceiver.java into an existing robot project.

After adding the file to the project, simply declare and define it in the `RobotContainer` class just like you would for a normal subsystem. The constructor takes one argument: the port to use to receive data from the Pi. This number should match the one that you set in your configuration in the app. The default for this is 3695.

```
public class RobotContainer {
  private final SubsystemReceiver SUB_KIWILIGHT = new SubsystemReceiver(3695);
```

Now, KiwiLight is fully configured in your robot project, and can be used to align your robot or its subsystems to a target. This can be done using a simple command running a PID loop. Use the `SubsystemReceiver` class's `getHorizontalAngleToTarget()` and `getVerticalAngleToTarget()` methods to run said PID loop.

```
@Override
public void execute() {
  if(kiwilight.targetSpotted()) {
    double horizontalAngle = kiwilight.getHorizontalAngleToTarget();
    double output = pidcontroller.calculate(horizontalAngle);
    drivetrain.driveDirect(output, output * -1);
  } else {
    drivetrain.driveDirect(0, 0);
  }
}
```

An example of how to use a `PIDController` to run a vision alignment loop using KiwiLight. `kiwilight` is a `SubsystemReceiver` object.

The following example will use a tank-style robot's drivetrain to align the robot to a target.

Example Project

This example project will demonstrate how to use a simple PID command to align a tank-style robot's drivetrain to a vision target using KiwiLight. Before following this project, ensure that you have implemented the `SubsystemReceiver` class into your robot code by following the instructions above ("How to add KiwiLight to your Robot").

First, define the drivetrain subsystem. The subsystem needs to have a method that sets the percent outputs of both sets of motors (left and right).

```
/**
 * Sets the percent outputs of the drivetrain motors. This method is required for
 * target alignment to work.
 * @param left Percent output to set the left motors to.
 * @param right Percent output to set the right motors to.
 */
public void driveDirect(double left, double right) {
  rightMaster.set(right);
  rightSlave.set(right);
  leftMaster.set(left);
  leftSlave.set(left);
}
```

Make sure your drivetrain subsystem is added to your `RobotContainer` as a subsystem.

```
public class RobotContainer {
  private final SubsystemDrive     SUB_DRIVE     = new SubsystemDrive();
  private final SubsystemReceiver SUB_KIWILIGHT = new SubsystemReceiver(Constants.KIWILIGHT_PORT);
```

Now, define a new `Command`.

After creating your `Command`, declare your drivetrain subsystem and a `SubsystemReceiver` object and define them using the command's constructor. The command should require the robot's drivetrain, but NOT the `SubsystemReceiver`. The `SubsystemReceiver`'s state is never changed.

```
public class CyborgCommandAlign extends CommandBase {
  private SubsystemDrive drivetrain;
  private SubsystemReceiver kiwilight;

  /**
   * Creates a new CyborgCommandAlign.
   */
  public CyborgCommandAlign(SubsystemDrive drivetrain, SubsystemReceiver kiwilight) {
    this.drivetrain = drivetrain;
    this.kiwilight = kiwilight;
    addRequirements(drivetrain);
  }
```

Then, declare a `PIDController` object, but do **not** define it in the constructor.

```
public class CyborgCommandAlign extends CommandBase {
  private SubsystemDrive drivetrain;
  private SubsystemReceiver kiwilight;
  private PIDController pidcontroller;
```

Now, in the command's `initialize()` method, define the `PIDController`. The kP, kI, and kD constants should be set using the dashboard's preference table. For simplicity's sake, we will use a method in our `Util` class called `getAndSetDouble()`, which is after the next image. This method simply returns the value of the preference and sets a backup if the preference does not exist. When writing your code, use your preference-grabber method in place of `Util.getAndSetDouble()`.

```
// Called when the command is initially scheduled.
@Override
public void initialize() {
  double kP = Util.getAndSetDouble("Align kP", 0);
  double kI = Util.getAndSetDouble("Align kI", 0);
  double kD = Util.getAndSetDouble("Align kD", 0);

  pidcontroller = new PIDController(kP, kI, kD);
}
```

For reference, the `Util.getAndSetDouble()` method is shown below.

```
static Preferences pref = Preferences.getInstance();
public static double getAndSetDouble(String key, double backup) {
    if(!pref.containsKey(key)) pref.putDouble(key, backup);
    return pref.getDouble(key, backup);
}
```

Then, in the `execute()` method, we will align the robot to the target. Firstly, it is very important that we check if a target is currently spotted. When no target is spotted, angles are reported as 180 degrees.

```
@Override
public void execute() {
  if(kiwilight.targetSpotted()) {

  }
}
```

In the `if` statement, use the horizontal angle reported by KiwiLight to calculate the percent output with which to turn. Set one side of the drivetrain to the output, and the other side to the inverse of that output (otherwise the robot would just drive forward)

```
@Override
public void execute() {
  if(kiwilight.targetSpotted()) {
    double horizontalAngle = kiwilight.getHorizontalAngleToTarget();
    double output = pidcontroller.calculate(horizontalAngle);
    drivetrain.driveDirect(output, output * -1);
  }
}
```

After that, add an `else` statement that sets the percent output of the motors to 0 if the target isn't seen:

```
@Override
public void execute() {
  if(kiwilight.targetSpotted()) {
    double horizontalAngle = kiwilight.getHorizontalAngleToTarget();
    double output = pidcontroller.calculate(horizontalAngle);
    drivetrain.driveDirect(output, output * -1);
  } else {
    drivetrain.driveDirect(0, 0);
  }
}
```

Then, fill out the `end()` and `isFinished()` methods. The `end()` method should set the percent output of the drivetrain motors to 0, and the `isFinished()` method should return `false` because the command does not really need to end.

```
// Called once the command ends or is interrupted.
@Override
public void end(boolean interrupted) {
  drivetrain.driveDirect(0, 0);
}

// Returns true when the command should end.
@Override
public boolean isFinished() {
  return false;
}
```

Then, in the `RobotContainer` class, link the command to a button on a controller, or send it to the dashboard to be displayed as a button.

```
private void configureButtonBindings() {
  CyborgCommandAlign align = new CyborgCommandAlign(SUB_DRIVE, SUB_KIWILIGHT);

  //linking the command to the A button on the driver's Xbox controller.
  //NOTE: The Xbox class is located in the Util folder. Xbox.A = 1.
  JoystickButton toggleAlign = new JoystickButton(DRIVER, Xbox.A);
  toggleAlign.toggleWhenPressed(align);

  //sending the command to the dashboard for display as a button
  SmartDashboard.putData("Align", align);
}
```

Now, you can push your robot code to the robot and test the alignment. However, before the robot will actually align to the target, the PID will need to be tuned. To do that, slowly increase the kP value from 0 until the robot steadily but accurately aligns to the vision target.

The full example project is located in KiwiLight's GitHub repository under the "KiwiLight - Rio Code" directory.

Class Reference

`public class SubsystemReceiver extends SubsystemBase`

A class that communicates with the coprocessor side of KiwiLight. This class receives and decodes data that KiwiLight sends it and provides a collection of methods that make accessing that data easy. Using this subsystem, programmers can write commands that align the robot (or parts of it) to retro-reflective targets.

`public SubsystemReceiver(int port)`

Creates a new `SubsystemReceiver`.

Arguments	
`int port`	The port to use to connect to the KiwiLight client. Before choosing a port, check the current year's game manual to ensure that the port is not blocked by FMS.

`public void periodic()`

Called periodically by the command scheduler. Updates the "Spotted" and "Updated" dashboard indicators.

`public boolean updated()`

Returns `true` if the subsystem has received a packet from KiwiLight in the last half second, `false` otherwise.

Returns: Whether or not the subsystem has recently received relevant data.

```
public double[] getLatestData()
```

Returns the latest data that was received from KiwiLight. The data is formatted as follows:
[0]: ID of target
[1]: X-coordinate of target (pixels from left of image)
[2]: Y-coordinate of target (pixels from top of image)
[3]: Width (pixels)
[4]: Height (pixels)
[5]: Distance
[6]: Horizontal Angle from Center (degrees)
[7]: Vertical Angle from Center (degrees)

Returns: Raw data received from KiwiLight. {-1, -1, -1, -1, -1,-1, 180, 180} is returned whenever no target is seen.

```
public double[] getSpottedTargetID()
```

Returns the spotted target's ID number, or -1 if no target is spotted.

Returns: ID of currently focused target.

```
public double[] getTargetPositionX()
```

Returns the spotted target's X position, or -1 if no target is spotted.

Returns: Target X coordinate in pixels from left of image.

```
public double[] getTargetPositionY()
```

Returns the spotted target's Y position, or -1 if no target is spotted.

Returns: Target Y coordinate in pixels from top of image.

```
public double getTargetWidthPixels()
```

Returns the width of the target in pixels, or -1 if no target is spotted.

Returns: Width of the currently seen target.

```
public double getTargetHeightPixels()
```

Returns the height of the target in pixels, or -1 if no target is spotted.

Returns: Height of the currently seen target.

```
public double getDistanceToTarget()
```

Returns the distance from the camera to the target. This value is reported in whatever unit it was configured in.

Returns: The distance from the camera to the target.

```
public double getHorizontalAngleToTarget()
```

Returns the number of degrees that the camera needs to be turned horizontally to be aligned to the target.

Returns: Horizontal angle to target, in degrees.

```
public double getVerticalAngleToTarget()
```

Returns the number of degrees that the camera needs to be turned vertically to be aligned to the target.

Returns: Vertical angle to target, in degrees.

```
public boolean targetSpotted()
```

Returns `true` if the target is spotted, `false` otherwise.

Returns: Whether or not KiwiLight is currently looking at a target.

```
public double getSecondsSinceUpdate()
```

Returns the number of seconds that have elapsed since the last packet was received from KiwiLight.

Tips and Troubleshooting

While setting up and using KiwiLight, it is possible that you may come across some problems. This chapter will cover a few potential problems that you may encounter, and discuss how to fix them. It will also offer some advice to help you quickly get KiwiLight working to the best of its ability.

Tips

- **Avoid powering the coprocessor with the robot battery.** The coprocessor should have its own battery to avoid sudden drops of voltage due to brownout, which may affect the performance of KiwiLight.
- **Use the recommended FRC camera settings.** FRC camera settings are optimal for seeing retro-reflective targets.
- **If possible and practical for your team, use a smart vision camera such as the JeVois A33.** These cameras allow you to program them to "preprocess" images before sending them to KiwiLight, and can increase KiwiLight's frame rate by as much as two times. If you are using a smart vision camera, make sure that you set the Preprocessor's "Type" setting to "Partial".

Troubleshooting

This table will cover some common issues that may be encountered while using KiwiLight, along with some potential fixes for those issues.

Issue	Fix
KiwiLight crashes frequently or freezes for unknown reasons, or does not start at all	Ensure that the coprocessor is completely up-to-date by running the commands `sudo apt-get update` and `sudo apt-get upgrade`
While running KiwiLight headlessly, communication with the robot from the Driver Station becomes unstable	This problem is most likely caused by KiwiLight packets overwhelming the robot router. Lower the "Send Cap" setting which is available in the Misc. tab in the KiwiLight program. This will lower the number of packets that are sent per second.
KiwiLight is supposed to start a	Sometimes, Cron does not allow KiwiLight to run

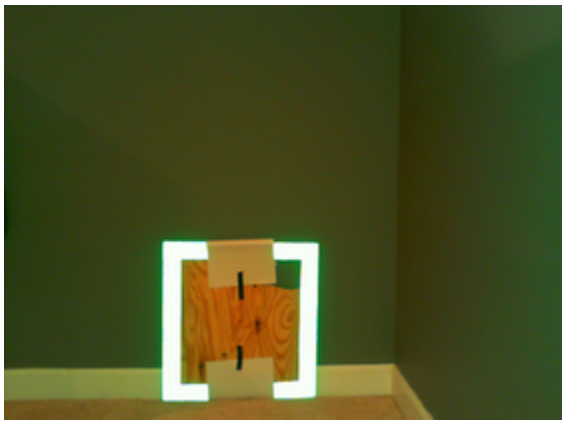
configuration automatically, but does not	when the Pi restarts. You may need to create a shell script that starts KiwiLight and then set the crontab to run that shell script automatically. To edit the crontab, use the command `crontab -e`. Then, in the editor that crontab creates, delete the line that starts KiwiLight automatically and replace it with `@reboot sh <full path to shell script>`.
KiwiLight cannot open a camera that is connected to the coprocessor	Ensure that no other instances of KiwiLight are running, and that the camera is not being used by another application. This problem is common if KiwiLight is scheduled to start automatically.
KiwiLight reports the same invalid data to the robot as if it has frozen	Check to make sure that the coprocessor running KiwiLight is still powered; a battery may have run out. If the coprocessor is being powered, use VNC to access the coprocessor's desktop and restart KiwiLight.
KiwiLight does not seem to aim the robot at the center of the target	Use the "Camera Offset" fields in the Misc. tab to adjust the robot center until the robot aims directly at the target.
Running automatically, KiwiLight sends data to the robot, but cannot see the target, even when the camera is looking directly at it	Restart KiwiLight, and make sure that the camera is connected to the coprocessor before KiwiLight starts again.

Algorithm

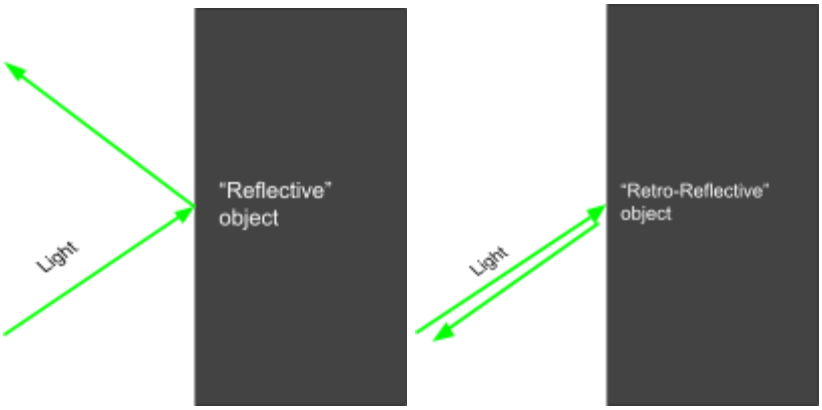
This chapter covers the algorithm that KiwiLight uses to search for retro-reflective targets. The algorithm is a version of the one described in the FRC Programming Done Right documentation, which can be found at frc-pdr.readthedocs.io/en/latest/. That documentation can be used for further reference.

In FRC, the computer vision challenges have almost always featured some sort of retro-reflective target. This allows teams to use a simple yet smart solution to solve the challenges. This section will cover the basics about how an efficient FRC vision algorithm works. More on the basic algorithm along with basic code that achieves it can be found at tinyurl.com/frcvisionalgorithm.

To illustrate how the algorithm works, let's suppose we have the following image that has been taken from a camera. Pictured in the image is our target of interest: the piece of wood with the retro-reflective brackets.

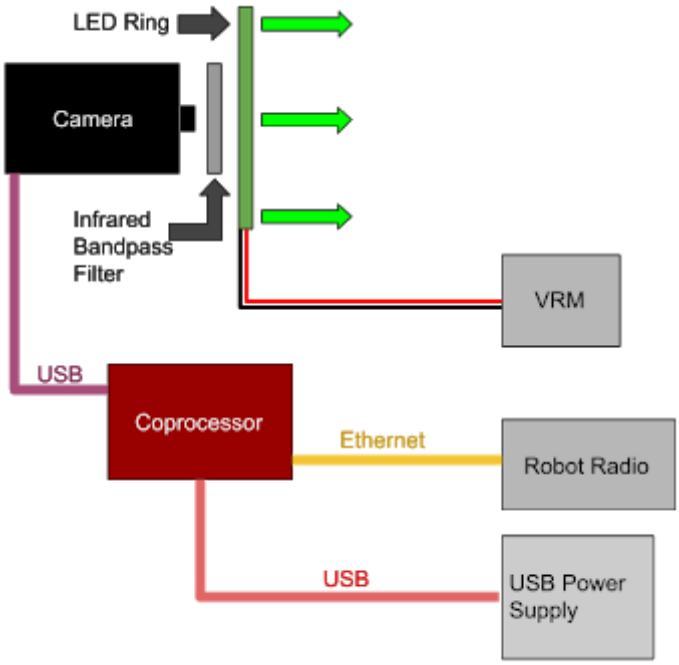


The retro-reflective aspect of the target is extremely important to the FRC vision algorithm. Below is an illustration of the difference between the properties of just "reflective" items versus the properties of "retro-reflective" items.



As seen in the image, light bounces off of a retro-reflective object directly towards the light source that it came from where objects that are just reflective do not. That being said, if we put the camera directly behind a light source such as an LED ring, any retro-reflective object will appear to the camera as a direct light source.

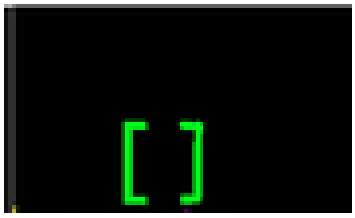
Now, we can look at the common FRC vision setup.



The camera should be located behind an LED ring which shines light towards the retro-reflective targets. For FRC applications, this LED ring should be green or infrared (but infrared is better), because neither of those colors are present on a typical FRC field, which is typically red and blue. The Infrared bandpass filter seen in the image should only be used if an infrared LED ring is used. This filter eliminates all colors except for infrared, ensuring that the camera sees as little noise from ambient light as possible.
The camera is connected to a coprocessor (computer other than the RoboRIO), which is typically a Raspberry Pi. The coprocessor will do the vision calculations and then send any target data to the RoboRIO over an Ethernet connection.

In order for the algorithm to work, the camera needs to be configured for retro-reflective targets. The camera's auto-exposure setting should be disabled, and its manual exposure setting should be set as low as possible. This ensures that the camera can only see direct light sources. In addition, the camera's auto white balance setting should be disabled. With the proper settings in place, images taken of our bracket target should look like this:

The next step is to apply a “threshold” to the image. This step tests the brightness of the pixels in the image. If a pixel is not above the “threshold,” then it is turned black. If it passes the threshold, then it is set to its maximum brightness.

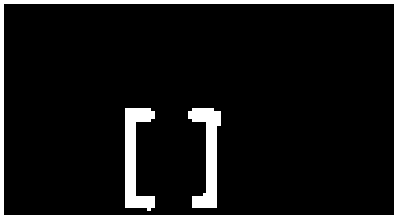


Next, we can optionally apply dilation and/or erosion to an image. Dilations take objects in an image and make them thicker. This can help to eliminate any imperfections that might be present in the image. Erosions make objects smaller, which can help to eliminate noise in an image.

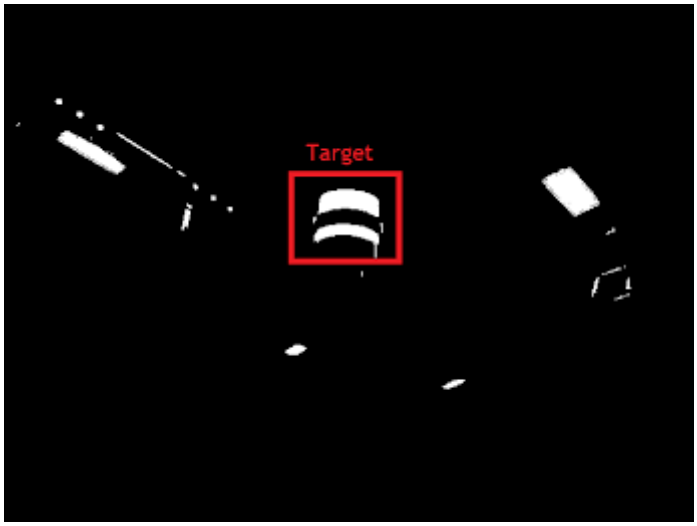
Left: undilated image | Right: dilated image.

Left: uneroded image | Right: eroded image.

Next, we eliminate all pixels in the image that are not green. By performing this “in-range” test, all pixels that are not in a range of possible shades of green are turned black. Those that remain are turned white. After the in-range test, the target might look like this:

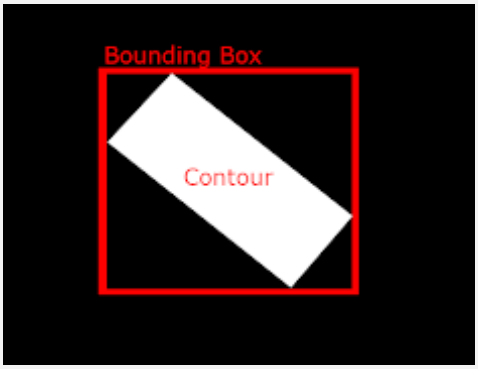


Now we need to look for the target itself. In a noiseless image like this one, that would be easy. Realistically however, the image probably has a few other objects, or “contours” in it. It will probably look more like this one from the 2017 FRC game:



Shown: Target from 2017 game, but with some undesired contours.

In order to eliminate the extraneous, invalid contours, you can perform a series of tests on specific contour properties. Contours whose properties are not within a range of acceptable values are eliminated. The table below shows a few of the different properties that contours can be tested on.

Property	Description
Area	The actual area of the contour itself.
Solidity	The ratio of the actual area of a contour to the area of its bounding box: $solidity = \frac{actual\ contour\ area}{bounding\ box\ area}$  That being said, the more space inside the bounding box the solid takes up, the greater this solidity value will be.
Angle	The angle of tilt of a contour's closest-fitting bounding box. This test was particularly useful for the 2019 game.
Aspect Ratio	The ratio of the width of the contour to the height of the contour:

	$aspect\ ratio = \frac{contour\ width}{contour\ height}$

The contours that remain are ideally the targets, or parts of targets, that we are looking for. If the target being searched for is a single contour, we can mark all remaining contours as targets. For targets with multiple contours, the only task remaining is to group the contours into the targets themselves, a task which may differ depending on the target's composition and contour count. An easy way to do this is to test contours' horizontal and vertical distances from each other against acceptable ranges of values.

Once the final targets are determined, information about the targets must be sent to the RoboRIO so that they can be used by the robot code. At the very least, the image coordinates of the targets should be sent. However, other good values to send include horizontal and vertical angles between the robot and the target, and the distance from the robot to the target. Data can be sent to the robot using a UDP (unreliable datagram protocol) or TCP (transmission control protocol) socket sender, which will send messages through Ethernet to the RoboRIO. Because your vision code will be sending multiple messages per second, a UDP socket is the best choice because it is the easiest to set up.