KiwiLight

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 the Raspberry Pi which uses generic USB webcams, as well as dedicated computer vision cameras, to solve FRC vision challenges at high framerates.

What you need:

While KiwiLight is capable of running on almost any Linux device, using almost any USB camera to do vision, Foximus Prime has found that some combinations work better than others. This our go-to vision setup:

- **JeVois A-33 Smart Machine Vision Camera**
    - https://www.jevoisinc.com/products/jevois-a33-smart-machine-vision-camera?variant=36249051018
    - Foximus Prime has also designed a 3d-printable mount for the JeVois which can be found here: https://www.thingiverse.com/thing:3521989
- **Raspberry Pi 4 Model B:**
    - https://www.raspberrypi.org/products/raspberry-pi-4-model-b/
- **60-80mm LED Ring:**
    - https://www.superbrightleds.com/cat/led-halo-rings/filter/Color,Green,136,4630:Diameter,80,181,7204:
    - NOTE: Can be any color, but for FRC green or infrared will be best (because the field is blue and red)
- **Portable USB Battery:**
    - Literally any phone charger. Must be able to provide 2-3 amps.
- **Ethernet Cable:**
    - Also will not include a link to this one because different teams have different robots that may require ethernet cables of different length.

Getting Started With JeVois (if using it):

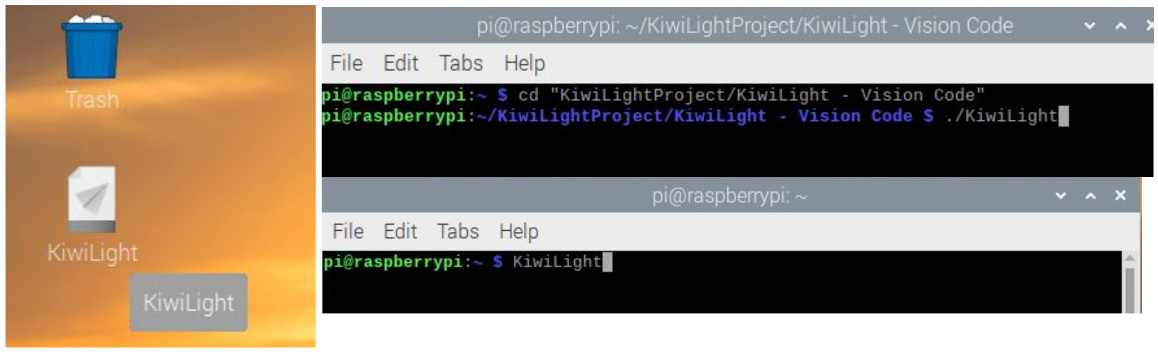
In FRC, vision has two parts: “Preprocessing,” and “Postprocessing.” “Preprocessing,” is where the image is taken and processed so that only the green chunks (or whatever color your LED ring is) can be seen. “Postprocessing” is where the green chunks are analyzed to see if they are targets. The JeVois camera has been very helpful in previous years because it can do the “preprocessing” part and send the preprocessed image to the Pi, meaning that the Pi only has to do the postprocessing. This helps to make vision faster and decrease latency.

- **Download JeVois Inventor.**
    - JeVois Inventor can be found at http://jevois.org/doc/JeVoisInventor.html
- **Load up some preprocessing code.**
    - Foximus Prime has some preprocessing code that is ready to be used and can be found at https://github.com/MSP1167/FoximusRobotics19/tree/d92f7eb0336683b6d87b95ad2e87696b82f988c0/VisionStuff, along with a step-by-step tutorial for how to load the code in for the first time. Note that since environments will be different, you will need to tune the code by adjusting the “threshold” value or the “lower” and “upper” colors based on the color of your LED ring.
- **Note the resolution of your module.**
    - Under the “Vision-Module” tab in Jevois Inventor, all modules should be listed with a resolution next to them. Note the resolution of your module, as it will be important later.

Getting Started With Raspberry Pi:

Now, let’s set up the Raspberry Pi. The Raspberry Pi will act as your “coprocessor,” so that the RoboRIO does not have to be burdened by vision calculations.

- **Download the latest version of KiwiLight.**
    - Releases can be found at https://github.com/wh1ter0se/KiwiLight/releases
    - If running on the Raspberry Pi, download “Raspbian.zip”
    - If running an Ubuntu computer, download “Ubuntu.zip”
    - If you are not running any of the above OSs, you will have to build KiwiLight from source. Download “Source Code (zip).”
- **Run the installer.**
    - Extract the folder you just downloaded anywhere you would like.
    - Launch the terminal and change into your extracted folder using: `cd <extracted directory>`
        - If building from source (NOT using Raspbian.zip or Ubuntu.zip): `cd “<extracted dir>/KiwiLight - Vision Code/”` (note the spaces in “KiwiLight - Vision Code”)
    - Enter the command: `sh install.sh`
        - If building from source, this may take a very long time as OpenCV also has to be built from source.
    - KiwiLight has been installed!
- **Launch KiwiLight.**

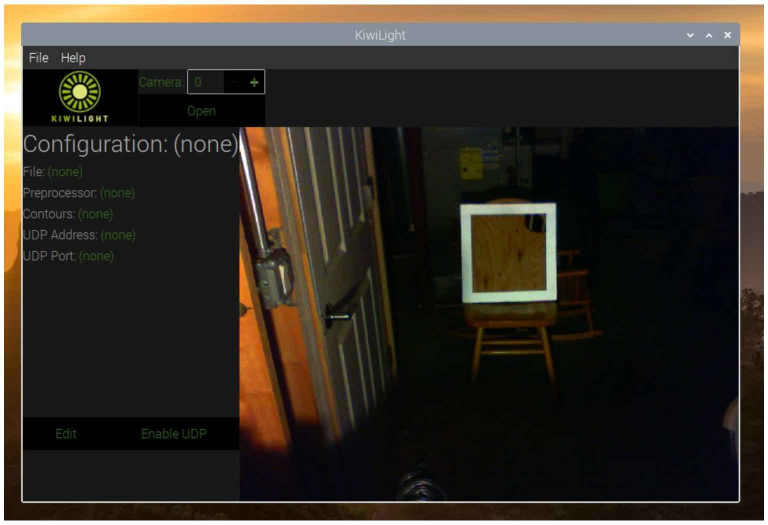


```
pi@raspberrypi:~ $ cd "KiwiLightProject/KiwiLight - Vision Code"
pi@raspberrypi:~/KiwiLightProject/KiwiLight - Vision Code $ ./KiwiLight
```

```
pi@raspberrypi:~ $ KiwiLight
```

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

- If the KiwiLight installer generated a desktop icon, double-click on that. Otherwise:
    - In terminal, enter the command `KiwiLight`
        - If you built KiwiLight from source: `cd` to where your installer was from above and enter `./KiwiLight`
- You should have a window that looks similar to this:



Camera: 0

Configuration: (none)

File: (none)

Preprocessor: (none)

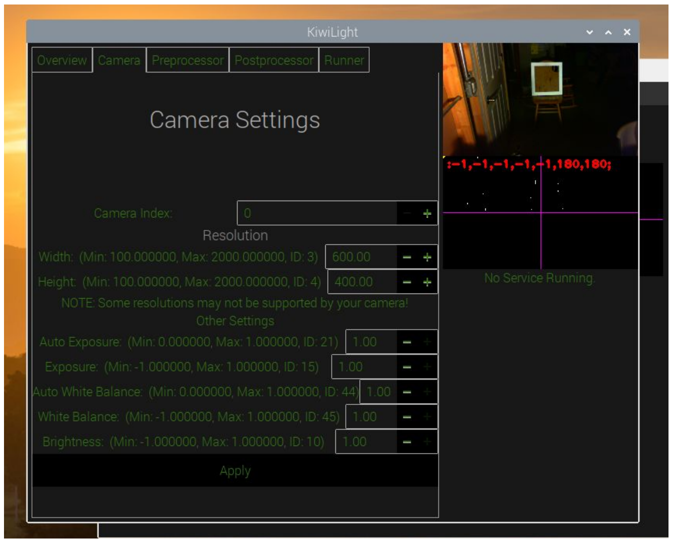
Contours: (none)

UDP Address: (none)

UDP Port: (none)

- **Create a new Configuration.**
    - Click “File” -> “New Configuration”

      OR
    - Click “Edit” when no configuration is loaded.
    - A window should pop up that looks like this:
- **Set your camera settings.**



Camera Settings

Camera Index: 0

Resolution

Width: (Min: 100.000000, Max: 2000.000000, ID: 3) 600.00

Height: (Min: 100.000000, Max: 2000.000000, ID: 4) 400.00

NOTE: Some resolutions may not be supported by your camera!

Other Settings

Auto Exposure: (Min: 0.000000, Max: 1.000000, ID: 21) 1.00

Exposure: (Min: -1.000000, Max: 1.000000, ID: 15) 1.00

Auto White Balance: (Min: 0.000000, Max: 1.000000, ID: 44) 1.00

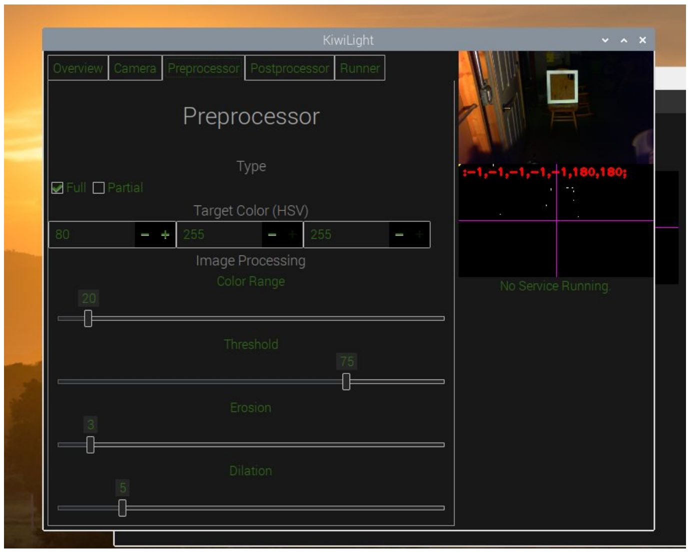
White Balance: (Min: -1.000000, Max: 1.000000, ID: 45) 1.00

Brightness: (Min: -1.000000, Max: 1.000000, ID: 10) 1.00

```
:-1,-1,-1,-1,-1,180,180;
```

No Service Running.

- Click “Camera” to enter the camera tab.
- If using JeVois, enter the resolution of your vision module into the “Width” and “Height” boxes, and hit “Apply.”
- If using some other camera, you will probably need to set other settings such as exposure, white balance, etc.
- **Configure Preprocessor.**



[✓ Full] Partial

Target Color (HSV): 80, 255, 255

Color Range: 20

Threshold: 75

Erosion: 3

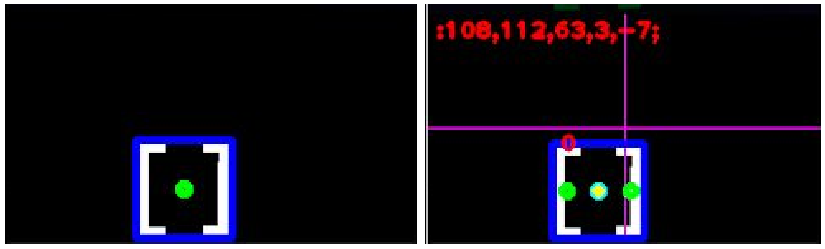
Dilation: 5

```
:-1,-1,-1,-1,-1,180,180;
```

- Click “Preprocessor” to enter the Preprocessor tab.
- If using JeVois, hit the “Partial” checkbox at the top (right under the big “Preprocessor” heading). You already did your preprocessor on the JeVois! Move on to the next step (“Learn the Target”).
- If not using JeVois, you will need to set the Preprocessor settings:
    - **Target Color**: The color of your LED ring, in HSV colorspace. The default for this is green.
    - **Color Range:** The amount of allowable error for all fields of the Target Color section.
    - **Threshold:** The amount of “contrast” to apply to the image. If you cannot see any white chunks in the output image (bottom right image), this value is probably too high.
    - **Erosion:** Used to eliminate noise by “eroding” it away. Be careful that you do not erode the entire target away though!
    - **Dilation:** Used to expand a target to fill any holes in it (make it a more solid shape). For reliable vision, this should be at least your erosion value plus 3.

- **Learn the Target.**
    - Click back into the “Overview” tab. At the bottom, click on the “Learn Target” button. You will be prompted to position the target in the center of the image, and ensure that it is hugged by a blue box. Make sure that there are no other “chunks” in the box or they will become part of the

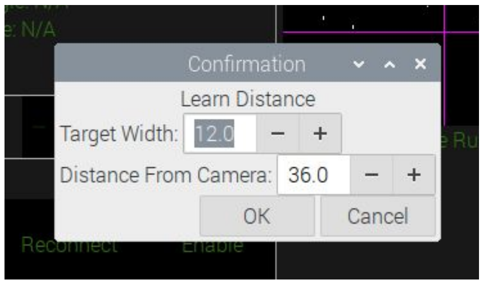
target! After you are done positioning the target, click “OK.” Once learning is done, the target should be surrounded by a blue box, with a light blue and yellow dot in the center. Each individual “object” in the target should also have a green dot on them.



Before learning(left), and after learning(right).

- If the target was not surrounded by a blue box before learning, then the object/objects were too small. Either move the target closer to the camera, or lower the “Minimum Area” property in the “Postprocessor” tab, and try again.

- **Learn the Distance Constants**



Confirmation

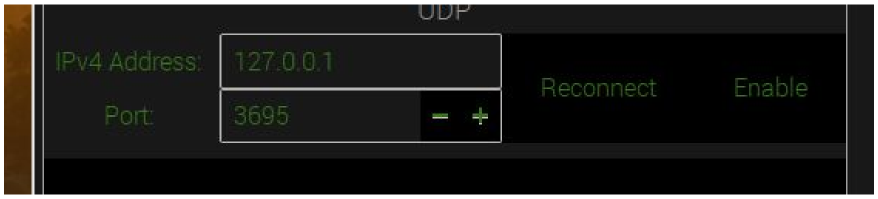
Learn Distance

Target Width: 12.0

Distance From Camera: 36.0

  - NOTE: This step, while kind of odd, ensures that the distance and angle values sent to the RIO are as accurate as possible.
  - Still in the “Overview” tab, click “Learn Distance,” next to the “Learn Target” button.
  - Measure the distance from the camera to your target, as well as the width of your target and enter into the confirmation box. These numbers can be in whatever unit you would like. The distance to the target will be reported in the same unit.
  - Before clicking “OK,” ensure that the target is seen by KiwiLight.
  - Click “OK.” When the learner is finished, you should see the “Target Distance” label in the Overview tab is the same number that you set as the “Distance From Camera” box.

- **Set up the UDP sender.**

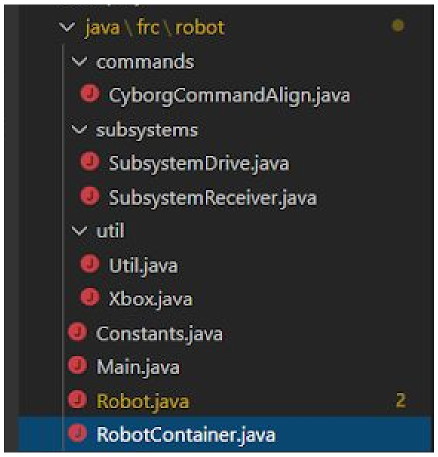


  - Still in the “Overview” tab (you can also do this in the “Runner” tab), enter the IPv4 address of the RoboRIO into the UDP “Address” field. The RIO’s IPv4 address can be found in the FRC Driver Station program in the “Communications” tab.
  - Set the UDP “Port” field to some port that is not blocked by FMS. The blocked ports can be found in the season’s game manual.
  - Hit the “Reconnect UDP” button.
  - To test your UDP settings, hit the “Enable” button next to the “Reconnect” button. If you have already set up KiwiLight in your robot code (see below), you should see the “Updated” dashboard indicator turn green.
- **Save your configuration.**
  - Still in the “Overview” tab, click “Save and Close.” if you are in a new configuration, a file dialog will pop up. Choose where you want to save your configuration and press “Save”

Adding the Robot Code:

Now that the Pi is processing vision, we need the RoboRIO to do things with it. A full robot project for robots with NEO drivetrains is provided at https://github.com/wh1ter0se/KiwiLight/tree/master/KiwiLight%20-%20Rio%20Code

- **Move SubsystemReceiver.java and Util.java Into your robot project.**

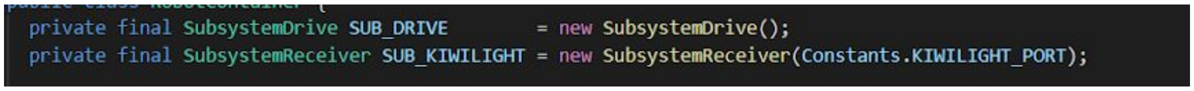


```
java \ frc \ robot
  commands
    CyborgCommandAlign.java
  subsystems
    SubsystemDrive.java
    SubsystemReceiver.java
  util
    Util.java
    Xbox.java
  Constants.java
  Main.java
  Robot.java                    2
  RobotContainer.java
```

(Note the files Util.java in the util folder and the SubsystemReceiver.java file in the subsystems folder)

- Copy SubsystemReceiver.java from the above link into your robot project (preferably into your "subsystems" folder). Make sure that the "package" heading at the top of the file is correct.
- Copy Util.java from the above link into your robot project (preferably into a folder called "util"). Also ensure that your package name is correct.

- **Instantiate SubsystemReceiver into your robot.**



```
  private final SubsystemDrive SUB_DRIVE        = new SubsystemDrive();
  private final SubsystemReceiver SUB_KIWILIGHT = new SubsystemReceiver(Constants.KIWILIGHT_PORT);
```

(RobotContainer.java)

- Wherever you define your subsystems in your robot code, add a line that declares a SubsystemReceiver object.

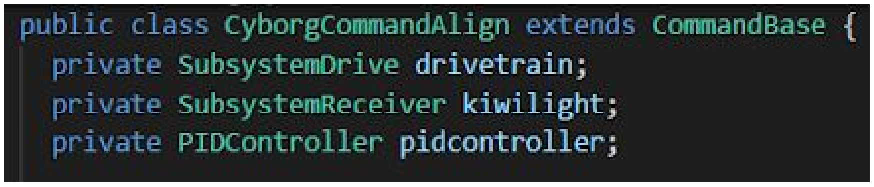
- **Note the helpful dashboard indicators.**
    - Open Shuffleboard (or SmartDashboard) and look for the three KiwiLight dashboard indicators: "Updated", "Spotted", and "rPi Data". Position them (or remove them) however you would like.
    - The indicators:
        - "Updated:" A BooleanBox. It is *true* when data has been received from KiwiLight within the last ½ second, *false* otherwise.
        - "Spotted:" Another BooleanBox. It is *true* when KiwiLight sees a target (rPi data is not "-1, -1, -1,180,180"), *false* otherwise.
        - "rPi Data:" A TextView. Shows the latest input string from KiwiLight. More about what the string means in the "What do all of those weird image markings mean?" section.

- **Write the PID.**
    - NOTE: For the PID, you can use almost any PID controller you want. In the past, Foximus Prime has had success using the WPILib PIDController class, the MiniPID class written by the Stormbots (FRC 2811), and the

TalonSRX's on-board PID Controller. For this part of the tutorial, we will use the WPILib PIDController class.

  - MiniPID Source code can be found at https://github.com/tekdemo/MiniPID-Java/tree/master/src/com/stormbots

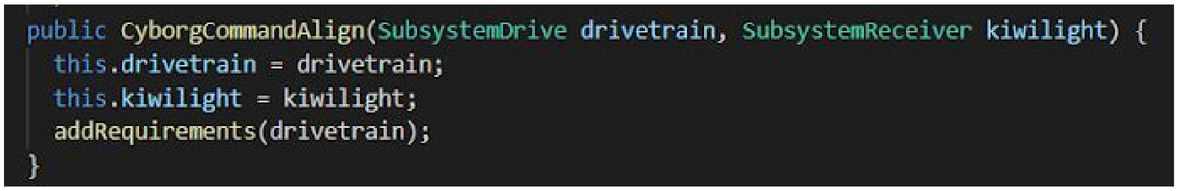
- Create a new command. This can be named anything you want, but in this tutorial it will be named "CyborgCommandAlign."
- In the new CyborgCommandAlign class, declare three objects. These can be named whatever you would like.
  - One SubsystemDrive (or whatever your drivetrain class is)
  - One SubsystemReceiver
  - One PIDController



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

(CyborgCommandAlign.java)

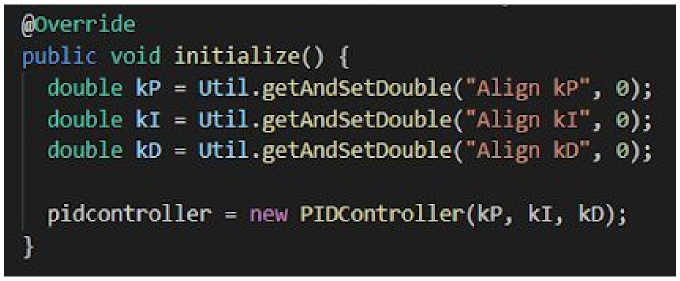
- Define the two new subsystems in the constructor, and add the drivetrain as a command requirement:



```java
public CyborgCommandAlign(SubsystemDrive drivetrain, SubsystemReceiver kiwilight) {
  this.drivetrain = drivetrain;
  this.kiwilight = kiwilight;
  addRequirements(drivetrain);
}
```

(CyborgCommandAlign.java)

- Define the PIDController in the initialize() method:
- NOTE: the Util.getAndSetDouble() method is defined in Util.java. It pulls numbers from the Preferences table on SmartDashboard/Shuffleboard. This means that the double kP will end up being whatever you set it to in Preferences.

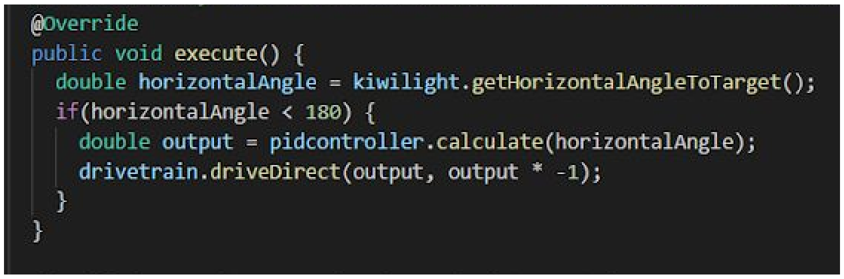


```java
@Override
public void initialize() {
  double kP = Util.getAndSetDouble("Align kP", 0);
  double kI = Util.getAndSetDouble("Align kI", 0);
  double kD = Util.getAndSetDouble("Align kD", 0);

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

(CyborgCommandAlign.java)

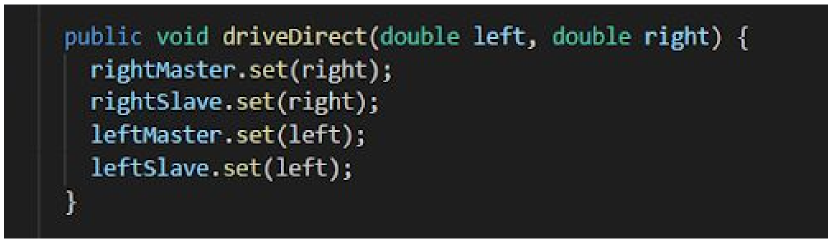
- Add the PID loop to execute(). This code will read the horizontal angle to target (kiwilight.getHorizontalAngleToTarget()), and then calculate and apply a PID output (if horizontalAngle < 180, which means that the target is seen).



```java
@Override
public void execute() {
  double horizontalAngle = kiwilight.getHorizontalAngleToTarget();
  if(horizontalAngle < 180) {
    double output = pidcontroller.calculate(horizontalAngle);
    drivetrain.driveDirect(output, output * -1);
  }
}
```

(CyborgCommandAlign.java)

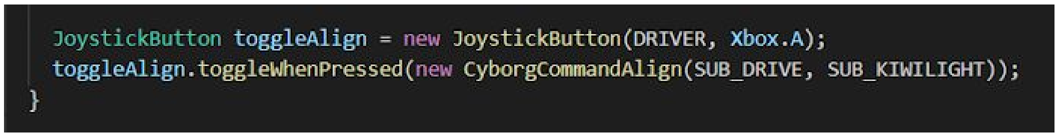
- NOTE: your driveDirect() method in your drivetrain subsystem should look like this:



```java
  public void driveDirect(double left, double right) {
    rightMaster.set(right);
    rightSlave.set(right);
    leftMaster.set(left);
    leftSlave.set(left);
  }
```

(SubsystemDrive.java)

- Link CyborgCommandAlign to a button in the RobotContainer class:
- NOTE: DRIVER is a private final Joystick object. Xbox.A is a constant that is defined in the Xbox class (Xbox.java is located in Util) It is not required to have the Xbox class, this is simply to demonstrate the linkage of a button to a command.



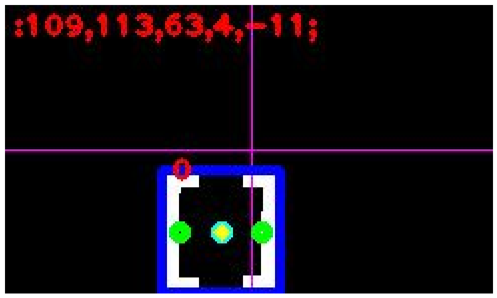
```java
  JoystickButton toggleAlign = new JoystickButton(DRIVER, Xbox.A);
  toggleAlign.toggleWhenPressed(new CyborgCommandAlign(SUB_DRIVE, SUB_KIWILIGHT));
}
```

(RobotContainer.java)

- Give it a test! Push the code, bring the target out in front of the camera, and press A (or whatever button the command is linked to). The robot should align with the target, and follow it when it is moved! Note that the PID will need to be tuned before the robot can follow the target reliably. PID values of all 0 will result in the robot not moving at all, and PID values that are too high can cause the alignment to be jittery (the robot might even overshoot the camera’s field of view).

What do all of those weird image markings mean?

- **The Output Image**



  - The KiwiLight output image gives you a lot of helpful information that you can use for debugging.
  - The red message in the top-left corner(":109,113,63,4,-11") is a direct output of the message that is sent to the RIO. You can also see this message on the "rPi data" indicator that SubsystemReceiver puts on the dashboard. Here's what the numbers mean:
    - The first number (109) is the x-coordinate of the target in image space.
    - The second number(113) is the y-coordinate of the target in image space.
    - The third number(63) is the distance (in whatever units you calibrate the distance in) between the camera and the target.
    - The fourth number(4) is the horizontal angle from the camera to the target in degrees. Positive angles are counterclockwise.
    - The fifth number(-11) is the vertical angle from the camera to the target in degrees. Positive angles are "up."
  - The pink crosshair in the center of the image is the estimated "robot center." The angles sent to the RIO are calculated relative to those lines. That is, if the yellow dot in the center of the target lies on top of the lines, the angles would be 0, no matter where the lines are in the image. The position of the lines can be changed using the "camera offset" settings in the Runner tab of the editor. NOTE that the lines will only appear offset if the target is seen.
  - The green dots sitting on top of the two brackets indicate that they can be a part of the target (If a target has two contours but only one is seen, that contour would have a green dot over it).