Using Generated Code in a Robot Program

GRIP generates a class that can be added to an FRC program that runs on a roboRIO and without a lot of additional code, drive the robot based on the output.

Included here is a complete sample program that uses a GRIP pipeline that drives a robot towards a piece of retroreflective material.

This program is designed to illustrate how the vision code works and does not necessarily represent the best technique for writing your robot program. When writing your own program be aware of the following considerations:

1. **Using the camera output for steering the robot could be problematic**. The camera code in this example that captures and processes images runs at a much slower rate that is desirable for a control loop for steering the robot. A better, and only slightly more complex solution, is to get headings from the camera and it's processing rate, then have a much faster control loop steering to those headings using a gyro sensor.
2. **Keep the vision code in the class that wraps the pipeline**. A better way of writing object oriented code is to subclass or instantiate the generated pipeline class and process the OpenCV results there rather than in the robot program. In this example, the robot code extracts the direction to drive by manipulating the resultant OpenCV contours. By having the OpenCV code exposed throughout the robot program it makes it difficult to change the vision algorithm should you have a better one.

Iterative program definitions

```
    package org.usfirst.frc.team190.robot;
import org.opencv.core.Rect;
import org.opencv.imgproc.Imgproc;
import com.ctre.CANTalon;
import edu.wpi.cscore.UsbCamera;
import edu.wpi.first.wpilibj.CameraServer;
import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.RobotDrive;
import edu.wpi.first.wpilibj.vision.VisionRunner;
import edu.wpi.first.wpilibj.vision.VisionThread;
public class Robot extends IterativeRobot {

        private static final int IMG_WIDTH = 320;
```

Using Generated Code in a Robot Program

```java
        private static final int IMG_HEIGHT = 240;

        private VisionThread visionThread;
        private double centerX = 0.0;
        private RobotDrive drive;

        private final Object imgLock = new Object();
```

In this first part of the program you can see all the import statements for the WPILib classes used for this program.

- The **image width and height** are defined as 320x240 pixels.
- The **VisionThread** is a WPILib class makes it easy to do your camera processing in a separate thead from the rest of the robot program.
- **centerX** value will be the computed center X value of the detected target.
- **RobotDrive** class takes encapsulates the 4 motors on this robot and allows simplified driving.
- **imgLock** is a variable to synchronize access to the data being simultaneously updated with each image aquisition pass and the code that's processing the corrdinates and steering the robot.

```java
    @Override
public void robotInit() {
    UsbCamera camera = CameraServer.getInstance().startAutomaticCapture();
    camera.setResolution(IMG_WIDTH, IMG_HEIGHT);

    visionThread = new VisionThread(camera, new Pipeline(), pipeline -> {
        if (!pipeline.filterContoursOutput().isEmpty()) {
            Rect r = Imgproc.boundingRect(pipeline.filterContoursOutput().get(0));
            synchronized (imgLock) {
                centerX = r.x + (r.width / 2);
            }
        }
    });
    visionThread.start();

    drive = new RobotDrive(1, 2);
}
```

Using Generated Code in a Robot Program

The **robotInit()** method is called once when the program starts up. It creates a **CameraServer** instance that begins capturing images at the requested resolution (IMG_WIDTH by IMG_HEIGHT).

Next an instance of the class **VisionThread** is created. VisionThread begins capturing images from the camera asynchronously in a separate thread. After processing each image, the pipeline computed **bounding box** around the target is retrieved and it's **center X** value is computed. This centerX value will be the x pixel value of the center of the rectangle in the image.

```
    @Override
public void autonomousPeriodic() {
        double centerX;
        synchronized (imgLock) {
                centerX = this.centerX;
        }
        double turn = centerX - (IMG_WIDTH / 2);
        drive.arcadeDrive(-0.6, turn * 0.005);
}
```

This, the final part of the program, is called repeatedly during the **autonomous period** of the match. It gets the **centerX** pixel value of the target and **subtracts half the image width** to change it to a value that is **zero when the rectangle is centered** in the image and **positive or negative when the target center is on the left or right side of the frame**. That value is used to steer the robot towards the target.