FINAL WEEK 9/30-10/7!

COMPUTER VISION

ROBO RAIDERS

SCARSDALE • 12331

What is CV (Computer Vision)?

Input Image

CV Pipeline

```
Transformations,
Filters,
etc.
```

Output Image

AND / OR

Output Data

```
{ large_edges: 42 }
```

>>> Computer vision is the act of using computers to convert an image into helpful data. <<<

Why do we need CV?

Purpose 1: To Direct Autonomous

Example

In the 2023-24 CENTERSTAGE Season, the autonomous would have to detect the state of a "team prop", and depending on the location of the team prop, the robot had to perform different actions.

This year, INTO THE DEEP uniquely is missing a "team prop detection" in the autonomous phase, but hopefully there will be one in future seasons!

Purpose 2: To Assist Driver Control

Example

PROBLEM: *In the 2023-24 CENTERSTAGE Season, drivers sometimes accidentally crashed into certain objects, thereby losing points.*

SOLUTION: *CV was used to detect when a robot was near a dangerous crash to slow or stop the robot before a point-deducting collision.*

CV REDUCES DRIVER WORKLOAD AND ERROR.

How do we use CV?

Utku's Super Cool Tutorial

https://scarsdale-robotics.gitbook.io/the-wiki/programming/opencv-tutorial-and-resources

I highly recommend you to follow Utku's tutorial from *sections one, then five through seven*. He will guide you through creating computer vision pipelines—a crucial prerequisite for using computer vision on the robot. Let the programming directors know if you have any questions!

(write your code after cloning this repository)

The PIXEL

Each image is composed of **pixels**. Each pixel is a single color.

Colors can be represented in different ways.

- **Pixels** in grayscale images each correspond to only one number, where 0 is black, 1 is white, and 0.5 is gray.
- **Pixels** in colored images take a few forms...

COLOR REPRESENTATIONS: RGB

Colored **pixels** are often represented by an **Red-Green-Blue (RGB)** value.

Each color is associated with a red value, a green value, and a blue value. Each such value is on a scale from **0** (minimum intensity) to **255** (maximum intensity).

(255, 0, 0)

(0, 255, 0)

(0, 0, 255)

(128, 50, 50)

(128, 50, 240)

COLOR REPRESENTATIONS: HSV

Colored **pixels** are often represented by a **Hue**-**Saturation**-**Value** combination.

Each color is associated with a hue from 0 to 359 (OpenCV is 0 to 179), a saturation from 0 to 255, and a value from 0 to 255. Saturation is “intensity” or “dullness”, and value is “brightness”.

(0, 255, 255)	(0, 128, 255)	(90, 255, 255)	(0, 255, 128)	(0, 128, 128)
High saturation and value	Lower saturation and high value	Furthest hue from red	High saturation, lower value	Lower saturation and value

Which is better?

PROS: **RGB**

Very common

Images from the camera come in RGB

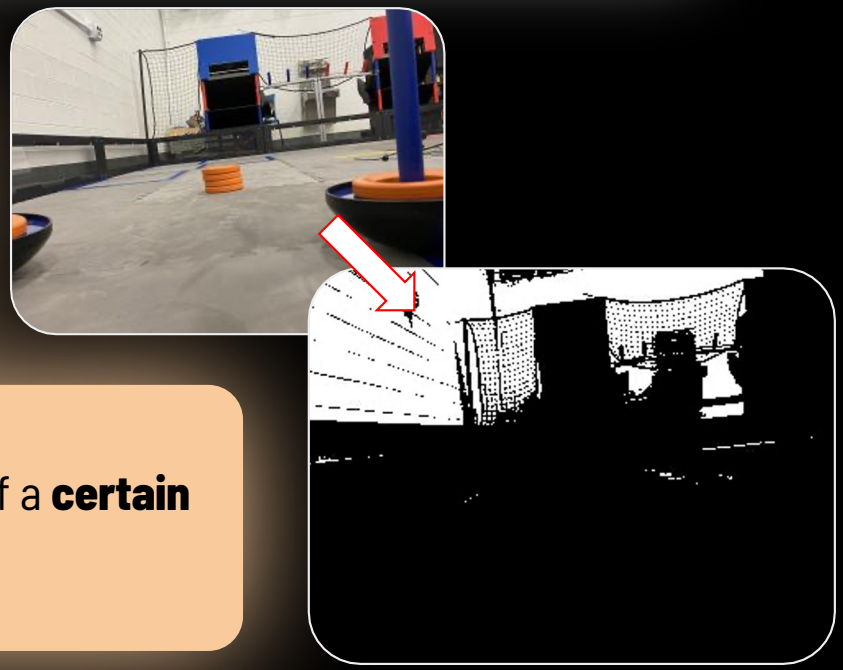
PROS: **HSV**

Resistant to lighting changes

We will use **HSV** for detecting colors because of the lighting-change resistance.

Color Thresholding

All CV code takes place in pipelines, as described in the first slide.



Color thresholding is the act of **finding** the **pixels** of a **certain color** in an image.

Color Thresholding

Input Image

CV Pipeline

- [1] Convert to HSV
- [2] Apply `inRange()`
- [3] Apply morphology
- [4] Convert to RGB
- [5] Apply mask
- [6] Draw contours

Output Image

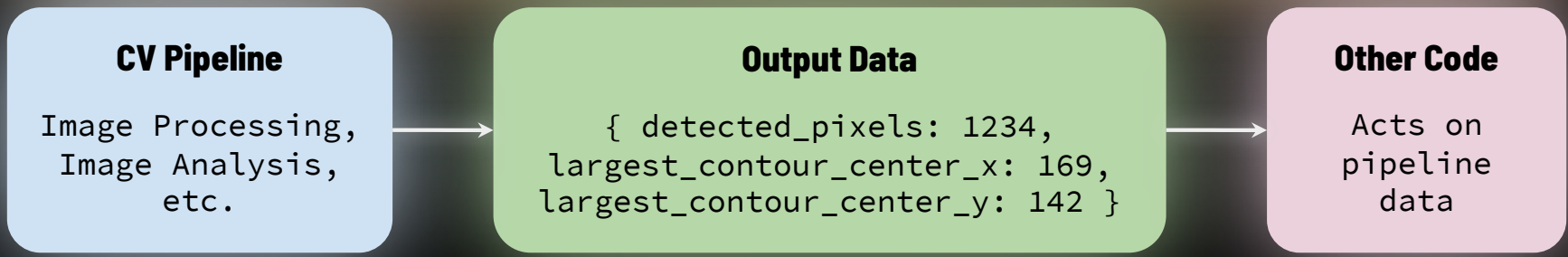
Make sure to read Utku’s tutorial for code!

Exporting Data from a CV Pipeline

In our robot, we want to **obtain** real-time information from **our computer vision pipeline**.

Therefore, we must **share** some information from **our pipeline** to **other code** (often to our OpModes).

Next pages of presentation: Code to create methods to share camera image information



The Code

In this example, we export two values from Utku's Step 7 code.

1. We export the number of detected (in-range) pixels.
2. We export the center of the bounding box of the largest contour.

https://github.com/Scarsdale-Robotics/2024-2025-Training-Program-U3/blob/master/TeamCode/src/main/java/org/firstinspires/ftc/teamcode/cvpipelines/testing/PipelineThatSharesInformation.java

OK... Good to Go?

Not yet! We need **one quick change** to run our pipeline on the robot.

Notice that all pipelines we have worked with so far extended the **`OpenCvPipeline`** class.

Although `OpenCvPipeline` objects work fine (and are needed for) the EasyOpenCV Simulator (EOCV-Sim), they are **not optimal for running pipelines on the robot**.

To ensure compatibility with the robot, we must convert our `OpenCvPipeline` to a **`VisionProcessor`** object.

The change is **surprisingly small**. You may even be able to figure it out by looking at the `VisionProcessor` equivalent of our previous code (next slide).

The Code

The pipeline that exported information, converted to a `VisionProcessor`.

I surrounded changes with comments in the below code.

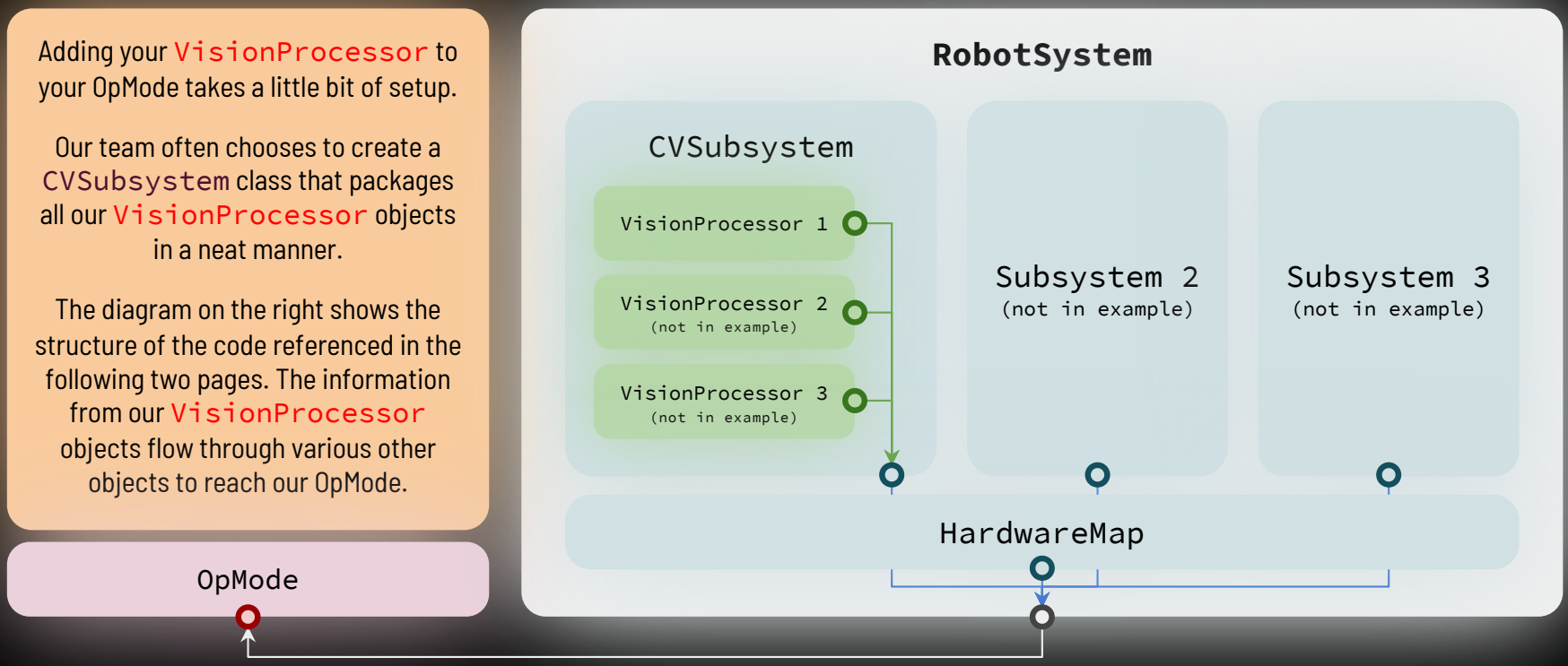
https://github.com/Scarsdale-Robotics/2024-2025-Training-Program-U3/blob/master/TeamCode/src/main/java/org/firstinspires/ftc/teamcode/cvpipelines/processors/PipelineThatSharesInformationAsVisionProcessor.java

Wait... But What About my OpMode?

Adding your `VisionProcessor` to your OpMode takes a little bit of setup.

Our team often chooses to create a `CVSubsystem` class that packages all our `VisionProcessor` objects in a neat manner.

The diagram on the right shows the structure of the code referenced in the following two pages. The information from our `VisionProcessor` objects flow through various other objects to reach our OpMode.



The `CVSubsystem` Code

This `CVSubsystem` contains supports one `VisionProcessor`—the one we made to detect the color orange and export the number of detected pixels, in addition to the center of the largest contour.

https://github.com/Scarsdale-Robotics/2024-2025-Training-Program-U3/blob/master/TeamCode/src/main/java/org/firstinspires/ftc/teamcode/subsystems/CVSubsystem.java

The OpMode Code

This OpMode references the `CVSubsystem`.

https://github.com/Scarsdale-Robotics/2024-2025-Training-Program-U3/blob/master/TeamCode/src/main/java/org/firstinspires/ftc/teamcode/opmodes/autons/AutonUsingCV.java