

# Generating Code from GRIP

## GRIP Code Generation

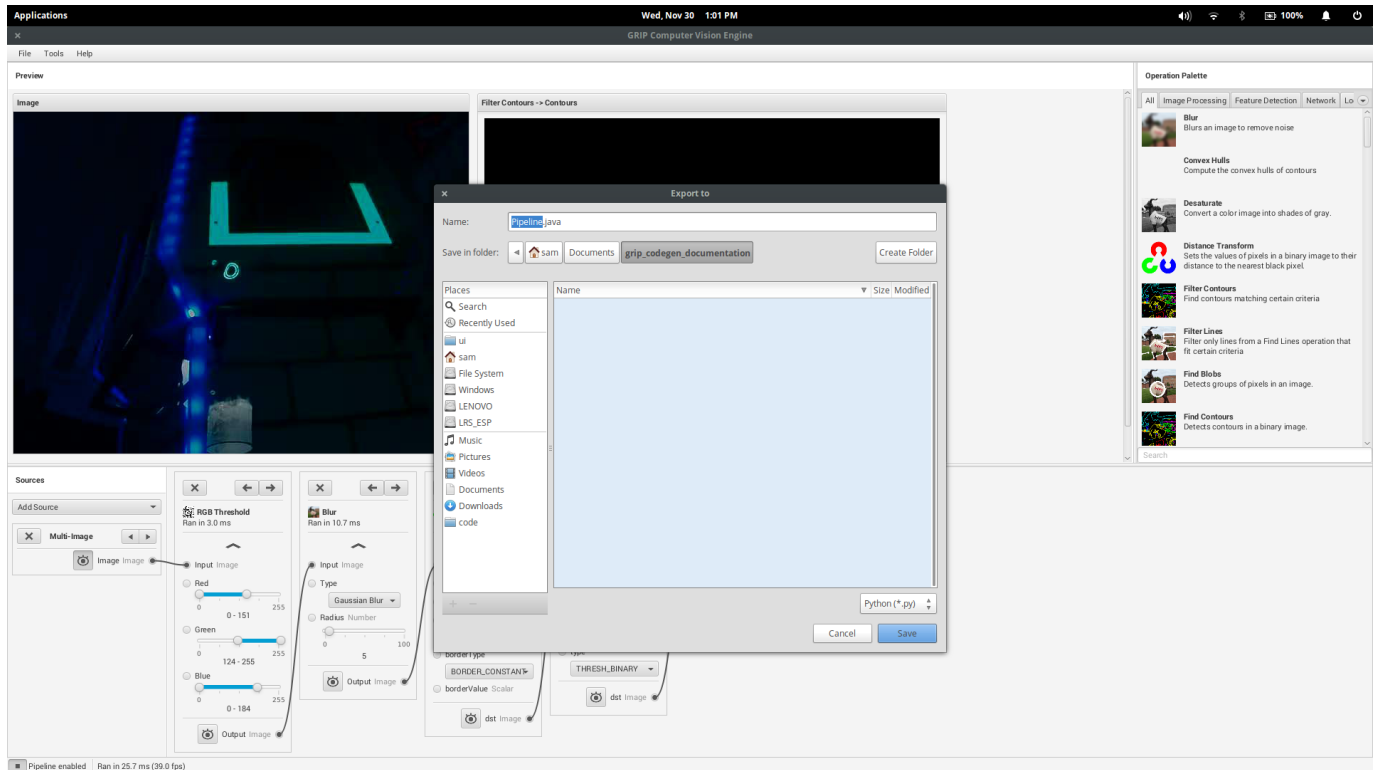
When running your vision algorithm on a small processor such as a roboRIO or Raspberry PI it is encouraged to run OpenCV directly on the processor without the overhead of GRIP. To facilitate this, GRIP can generate code in C++, Java, and Python for the pipeline that you have created. This generated code can be added to your robot project and called directly from your existing robot code.

Input sources such as cameras or image directories and output steps such as NetworkTables are not generated. Your code must supply images as OpenCV mats. On the roboRIO, the CameraServer class supplies images in that format. For getting results you can just use generated getter methods for retrieving the resultant values such as contour x and y values.

## Generating Code

To generate code, go to **Tools > Generate Code**. This will bring up a save dialog that lets you create a C++, Java, or Python class that performs the steps in the GRIP pipeline.

# Generating Code from GRIP



If generating code to be used in a pre-existing project, choose a relevant directory to save the pipeline to.

**C++ Users:** the pipeline class is split into a header and implementation file

**Java Users:** the generated class lacks a package declaration, so a declaration should be added to match the directory where the file was saved.

**Python Users:** the module name will be identical to the class, so the import statement will be something like `from Pipeline import Pipeline`

## Structure of the Generated Code

```
Pipeline:
// Process -- this will run the pipeline
process(Mat source)
```

# Generating Code from GRIP

```
// Output accessors
getFooOutput()
getBar0Output()
getBar1Output()
...
```

## Running the Pipeline

To run the Pipeline, call the process method with the sources (webcams, IP camera, image file, etc) as arguments. This will expose the outputs of every operation in the pipeline with the `getFooOutput` methods.

## Getting the Results

Users are able to the outputs of every step in the pipeline. The outputs of these operations would be accessible through their respective accessors. For example:

Operation	Java/C++ getter	Python variable
RGB Threshold	<code>getRgbThresholdOutput</code>	<code>rgb_threshold_output</code>
Blur	<code>getBlurOutput</code>	<code>blur_output</code>
CV Erode	<code>getCvErodeOutput</code>	<code>cv_erode_output</code>
Find Contours	<code>getFindContoursOutput</code>	<code>find_contours_output</code>
Filter Contours	<code>getFilterContoursOutput</code>	<code>filter_contours_output</code>

If an operation appears multiple times in the pipeline, the accessors for those operations have the number of that operation:

Operation	Which appearance	Accessor
Blur	First	<code>getBlur0Output</code>
Blur	Second	<code>getBlur1Output</code>

## Generating Code from GRIP

Operation	Which appearance	Accessor
Blur	Third	<code>getBlur2Output</code>