April 2016

GRIP: Graphically Represented Image Processing engine

Jonathan L. Leitschuh
Worcester Polytechnic Institute

Thomas John Clark
Worcester Polytechnic Institute

GRIP: Graphically-Represented Image Processing

A Major Qualifying Project submitted to the faculty of

Worcester Polytechnic Institute

in partial fulfillment of the requirements
for the degree of Bachelor of Science

April 28, 2016

By:
Thomas J. Clark
Jonathan L. Leitschuh

Submitted To:
Professor Michael Gennert
Professor Brad Miller
Worcester Polytechnic Institute

Abstract

The goal of this project was to build an application that non-experts could use to construct computer vision algorithms, and more experienced users could use to develop algorithms faster. Given the time constraints imposed by robotics challenges like the *FIRST* Robotics Competition, computer vision is often underutilized by even the most experienced players. We used Java and OpenCV to implement a user interface to make rapidly developing vision systems easier. As a result, many teams successfully used our software in the 2016 *FIRST* Robotics Competition. We believe that our application could be used for further applications in research and education.

Contents

List of Figures

1 Introduction

Computer vision (CV) is a field in computer science that deals with the processing of visual data into abstract information, such as the location of an object. Computer vision is applicable to computer science and robotics in particular, although CV techniques can be appropriate for any problem that can be thought of as automatically extracting high-level details from images.

Computer vision solutions typically exist as a pipeline of data. The pipeline's input consists of images and the final output is the desired high-level feature data. Each step of the pipeline extracts information into a representation suitable for analysis by the next step. The building blocks of computer vision are simple, but in practice constructing a pipeline requires choosing from many different possible operations and fine-tuning the parameters to each one. This can make it a difficult field to begin in without either having a solid background in CV theory or many iterations of experimentation. Unfortunately, the most popular existing toolkits are not ideal for quickly experimenting, and areas like competitive robotics and academic projects do not allow a large amount of time for experimenting.

This problem has been previously acknowledged, and several open-source projects and commercial products exist in response to it. OpenCV (Open Source Computer Vision Library) is a comprehensive software library that contains over 2,500 operations. It is used extensively by companies and researchers [1]. RoboRealm is a proprietary Windows application that aims to simplify the process of computer vision programming using a point-and-click interface [2]. A similar tool is NI Vision Assistant, which helps create computer vision algorithms without programming by automatically generating LabVIEW or C code.

While OpenCV is the most complete and widely-used of these solutions, it is not suitable for beginners to experiment and prototype. OpenCV is a software library, so any change to the pipeline or any parameters requires editing the source code, recompiling, and running the program again. In addition, getting started with OpenCV requires knowledge of a supported programming language and the OpenCV API, which makes it an inconvenient choice for beginners to computer vision. For more experienced users, OpenCV is a more viable option, but as a programming API it still requires recompiling and running code whenever a parameter or step of an algorithm changes. Even veteran computer

vision programmers could benefit from a more streamlined development process. RoboRealm improves this workflow by providing a user interface for selecting algorithms and tuning parameters. However, most users must purchase a license to use RoboRealm, and the source code is not available, making it less attractive as an option for students trying to learn computer vision. In addition, there is potential for a more intuitive user interface than the one in RoboRealm.

To fill this gap, we created GRIP (Graphically-Representing Image Processing engine). GRIP is a program that beginners can use to solve computer vision problems without any prior expertise with the field or with programming in general. GRIP can be used to create real algorithms that can be deployed, for example, as part of a robotic control system, or just for educational, research, and prototyping use. We designed the initial version of GRIP with *FIRST* Robotics Competition in mind because FRC teams falls within one of our intended audiences — novice programmers seeking to solve a computer vision challenge with limited time — we intend for GRIP to improve the workflow of computer vision users of any experience level.

2 Background

2.1 Computer Vision Fundamentals

Computer vision is the use of algorithmic techniques to understand images of the real world. It is important to many fields, such as driver assistance systems, industrial automation, and consumer products like mobile phones. [3, p. vii] A single computer vision application involves a combination of many techniques, which fall into several broad categories.

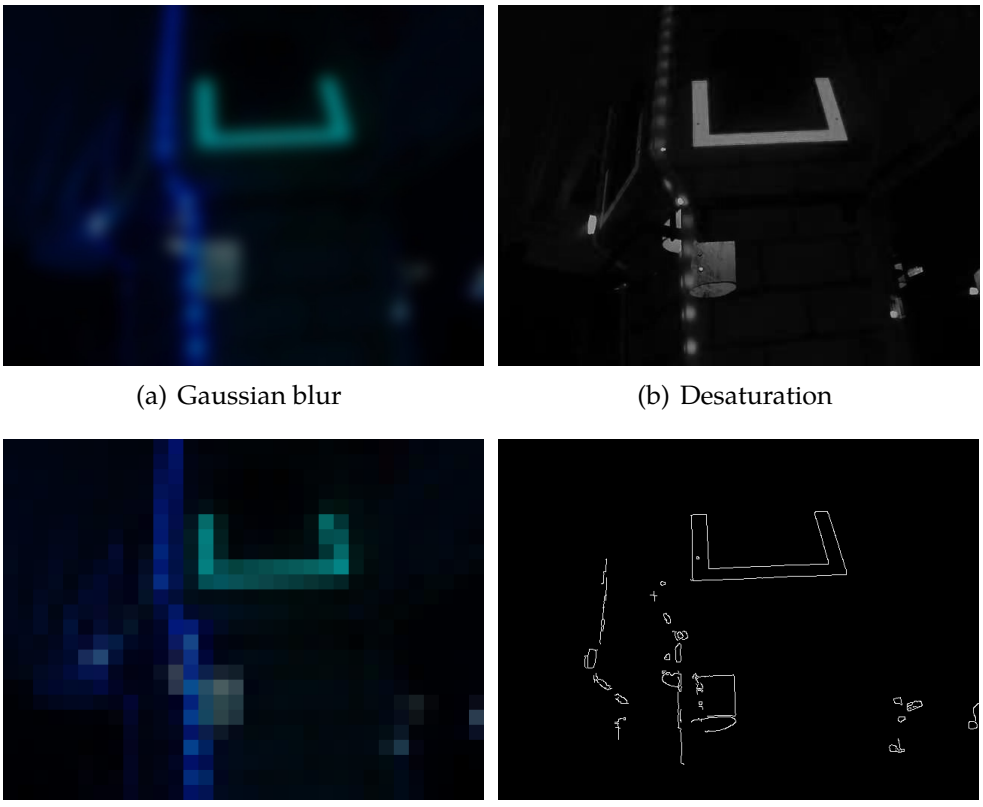
Image acquisition is the most important step in implementing practical computer vision systems, since shortcomings in acquiring initial inputs are likely to result in flawed analysis from subsequent steps. One critical aspect of this is proper lighting. In order for algorithms to analyze an object, the image must have enough contrast to distinguish the boundaries between different objects, as well as consistent enough lighting to recognize which areas of the image are part of the same object. One technique for achieving uniform lighting is a ring-shaped light source around the camera, which illuminates the sides of any objects that face the camera. [4, ch. 27]

The next step after image acquisition is image processing. The main goal of processing images in computer vision is to remove some information, such as lighting, textures, or background objects, leaving only information that's useful for extracting data. An example of this would be edge detection — all of the colors are removed from an image, since it's much more useful to know boundaries between objects than every single pixel of every object. Image processing also includes techniques for scaling an image to a smaller resolution for faster computation, transforming an image into grayscale, and blurring an image to remove noise. [3] Figure 1 shows examples of Gaussian blur, desaturation, bicubic interpolation, and Canny, which are specific techniques for tasks such as blurring, resizing, and edge detection.

Image segmentation is typically performed next in computer vision. Image segmentation refers to partitioning the pixels of an image into groups of connected pixels based on a similar quality they share. The most common technique is by applying a threshold, where connected pixels within a certain range of hue, saturation, lightness, or other values are grouped in a segment. This type of segmentation results in a binary image,

since every pixel has two possible states - within the threshold or not within the threshold.

Next, some sort of feature extraction is done on the processed image. In this step, algorithms are used that attempt to recognize shapes and patterns in an image that might indicate some object of interest. A simple example of this is contour detection, where a two-dimensional binary segmented image is transformed into lists of points forming the boundaries around each segment. These lists can then be filtered by qualities like their area or their shape to eliminate all data except the object of interest. While raw two-dimensional images are easy for humans to interpret, high-level feature data like this is much easier for computer programs to act on, since it reduces a complicated, noisy, real-world view into a few numbers that represent the important parts of the image.



(a) Gaussian blur (b) Desaturation

(c) Bicubic scaling (d) Canny edge detection

Figure 1: An image with various image processing operations applied

Lastly, the final phase of a computer vision process is to make a decision based on the data gathered from the image. This step is obviously very different for different computer vision applications, and could include actions as diverse as turning a robot to face towards a target, discarding a defective manufactured product, or steering a self-driving car to the proper location on the road. Unlike the previous steps, this one isn't composed of a set of widely accepted algorithms. Therefore, computer vision solutions such as OpenCV and GRIP only focus on the first few steps, and it's up to the author of the decision-making code to write this one.

2.2 Existing Solutions

2.2.1 OpenCV

OpenCV is an extremely popular computer vision software library, used extensively in industry, research, and by hobbyists. The library provides over 500 computer vision algorithms that can be used by programmers working in C, C++, Python, Java and MATLAB on multiple operating systems. [1]

2.2.2 NI Vision Assistant

NI Vision Assistant is a commercial application for creating image processing algorithms. Vision Assistant includes a GUI that can be used to acquire images, apply image processing filters, and analyze images to find high-level features.

Unlike RoboRealm, NI Vision Assistant algorithms are deployed by generating source code. Vision Assistant generates code that uses National Instrument's proprietary computer vision API in several different programming environments, including LabVIEW, .NET and C [5]. A major advantage of NI Vision Assistant's code generation is portability — algorithms can run on any device supporting NI Vision, even FPGAs. However, this means that either the final decision-making steps of the control system must be written in one of the supported programming languages, or some application-specific communication protocol must be written to relay vision data from the generated code to another program. This constraint could make NI Vision Assistant inconvenient to use for devel-

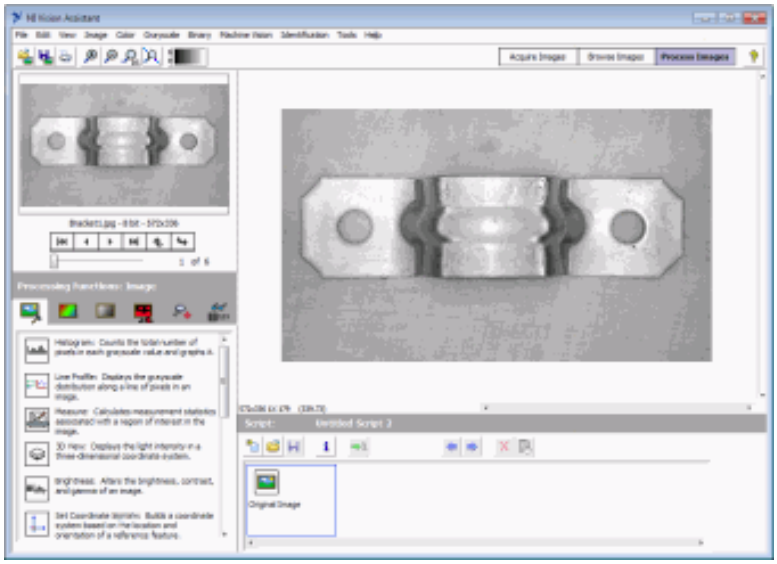


Figure 2: NI Vision Assistant [5]

opers working with common programming languages like Python or Java. Because the generated code uses the commercial NI Vision library, the potential applications are also limited to machines with a valid license, which could be prohibitively expensive for hobbyists and students.

2.3 Researchers

Robotics research combines a variety of fields such as electrical, mechanical and computer science. Robotic researchers may have a depth in one particular field but not in others. A project may need a fast solution to a software challenge that is not the primary concern of the research. Currently, the easiest way to solve many of computer vision challenges in the research domain is using OpenCV.

2.3.1 Robot Operating System (ROS)

"The Robot Operating System (ROS) is a flexible framework for writing robot software. It is a collection of tools, libraries, and conventions that aim to simplify the task of creating complex and robust robot behavior across a wide variety of robotic platforms" [6]. ROS

provides the infrastructure researchers need to quickly solve a robotic challenge using packages that other developers have created to perform specific robotic computations like inverse kinematics, driving, or performing SLAM (Simultaneous localization and mapping). ROS is designed to use a publisher and subscribers model to allow message to be sent between different processes and across networks. This allows robotic programs to interact agnostic from the language or OS that is running. The ROS platform provides the full stack of development requirements. It includes a build system to implement your own source code, a custom way to define messages that are passed between nodes, a dependency manager that allows you to pull in other developers projects, and the run environment to allow these various programs to communicate. The preferred OS for ROS development and running ROS programs is Ubuntu. This created a problem for GRIP because one of our goals was to be agnostic of the operating system and environment.

Because ROS is already used extensively within the robotic research domain, it is a promising protocol for third-party tools targeting researchers to support. A program using ROS could run on the research robot's OS, or the master OS and ROS robot programs would be able to communicate with it without requiring additional software libraries or unfamiliar APIs.

2.4 FIRST Robotics Competition

FIRST Robotics Competition (FRC) is an annual robotics challenge organized by *FIRST* (For Inspiration and Recognition of Science and Technology). FRC teams typically consist of around twenty high school students who are guided by volunteer mentors. Each year, students and mentors spend a six-week build season designing, building, and programming a robot to compete against other teams in their district and in the *FIRST* Championship. [7]

Each year's game, which is announced at the beginning of build season, is different from the previous games. For example, the 2015 game, Recycle Rush, required teams to design robots that would cooperate to stack plastic totes and recycling bins. [8] The 2016 game, *FIRST* Stronghold, consists of robots navigating over obstacles to score balls in goals on their opponents' towers. [9] Because of these differences, FRC teams must develop both a strategy and technical design essentially from scratch within the six week build season.

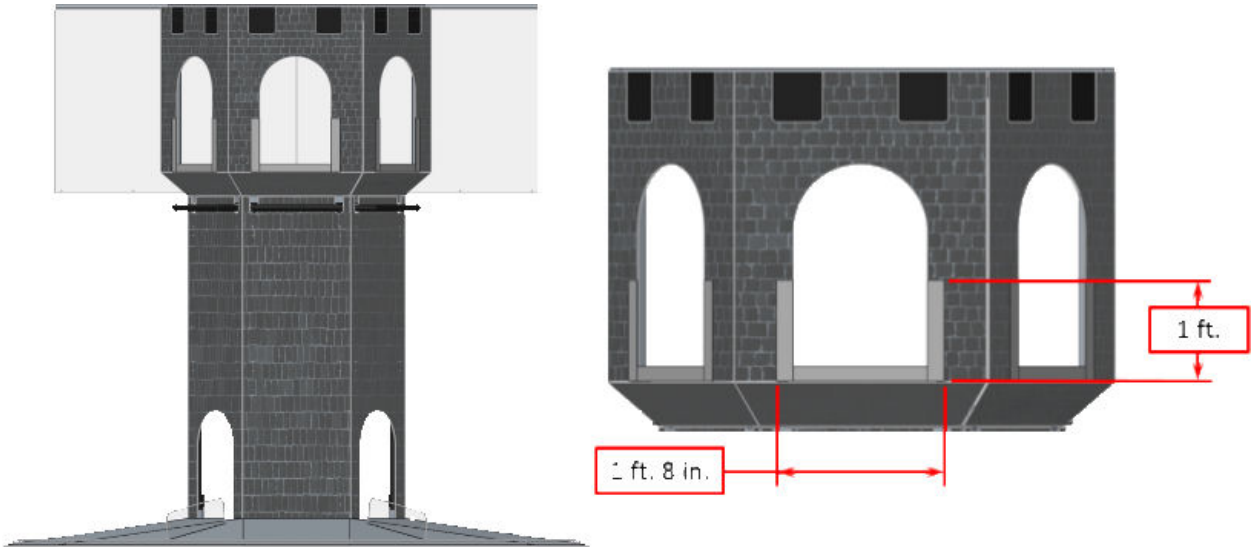


Figure 3: *FIRST* Stronghold tower drawings [8]

2.4.1 Computer Vision in FRC

One shared component of different *FIRST* Robotics Competition games is an autonomous period where robots operate without human drivers, instead relying on sensor data and pre-programmed instructions to score bonus points. *FIRST* Stronghold, for instance, includes a 15-second autonomous period at the beginning of each match where robots may score in their opponents' tower goals for 10 points each instead of 5. [9] Because this must be done without the assistance of human players, computer vision is an attractive technique for reliably completing this portion of the challenge. Computer Vision is also important in the rest of the match, called the teleoperated period, where human drivers may use computer-aided automatic targeting for faster scoring.

To make the use of computer vision easier, the *FIRST* Stronghold goals are surrounded by a U-shaped arrangement of retro-reflective tape, as shown in Figure 3. The tape reflects light back towards its source, so a ring of LEDs around a camera lens will cause the camera to pick up an intense brightness that can be used to detect the goal. Figure 4 shows this effect combined with a very small exposure time, which causes everything but the brightest objects in the frame to appear black. This setup is well-suited for computer vision processing because the goal can be cleanly segmented from the background by simply comparing the brightness of each pixel to a certain range.

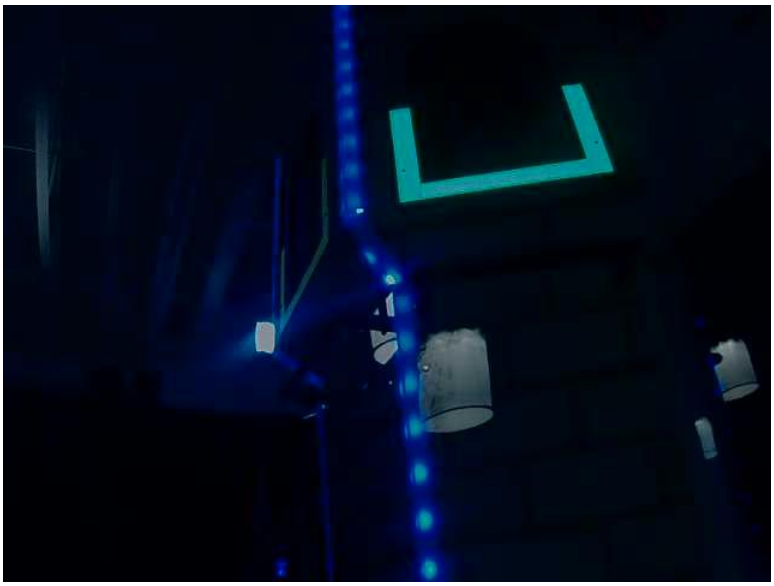


Figure 4: *FIRST* Stronghold sample image, captured with LED ring and low exposure

2.4.2 NetworkTables

NetworkTables is a network protocol for communicating key-value pairs between multiple clients, designed specifically for FRC. [10] Software libraries to access NetworkTables are included in the distribution provided to FRC teams, and it is the de facto standard for communication in FRC robot control systems.

Clients can use the protocol to send key-value entries to a server running on the robot controller, which then notifies all other clients. Keys are conventionally slash-separated strings, such as `"SmartDashboard/rotation/z"`. Values can be booleans, numbers, strings, arrays of any of the above, or raw data.

NetworkTables is used by robot controllers, driver station computers, and other hardware on the same network for scenarios such as:

- Sending real-time sensor data from the robot controller to a dashboard program for viewing by drivers or programmers.
- Sending debug messages from the robot controller to the dashboard.
- Sending configuration options from the driver station to the robot controller. This allows teams to include multiple autonomous programs that could be selected before

a match without reloading the robot's software.

- Sending the result of a computer vision algorithm from a custom team-made program to the robot controller.

Because NetworkTables is already used extensively within the FRC control system, it is a promising protocol for third-party tools supporting FRC teams. A program using NetworkTables could run on the robot controller, on the driver station, or on another piece of hardware, and FRC robot programs would be able to communicate with it without requiring additional software libraries or unfamiliar APIs.

The disadvantage of NetworkTables is that it is not commonly used outside of FRC. Researchers, students, and hobbyists working outside of the *FIRST* ecosystem would be less likely to write software to integrate with programs unless they support other more general-purpose protocols.

3 Design and Methodology

3.1 Goals

The goal of this project was to create a tool for computer vision that could help both newcomers to the field and experienced users rapidly develop algorithms. To implement this, we developed GRIP with the following requirements:

- The user interface must be intuitive to use, so users can discover for themselves how to construct computer vision algorithms. To judge the completion of this requirement, we interacted with FRC teams who used GRIP, many of which were made up of high schoolers with only some preliminary computer programming experience.
- The application must support a useful number of computer vision operations. This is somewhat ambiguous alone, so we used *FIRST* Stronghold as a use-case to prove that GRIP could be used to solve a moderately complicated computer vision challenge.
- The application must be able to run a pre-made pipeline in a "headless" mode without a user interface, so a pipeline can be developed on a desktop computer and deployed to an embedded device. We used the FRC controller, the roboRIO, to test this feature, with the expectation that it could also work on similar devices used outside of FRC.
- The application must run on most common operating system and computer architectures to reach the largest possible audience. Specifically, we made sure GRIP runs on Windows, OS X, and several Linux distributions.
- The application must be open for anyone to modify, so in the future people and companies can add features we didn't think of.

3.2 Project Schedule

The initial planning phase for GRIP begin in D term of 2015. Because *FIRST* Robotics Competition was such an important initial use case, our schedule was designed so that an early version of the program would be available to FRC teams before the beginning of build season, and the first stable release would correspond roughly with the beginning of build season. Throughout the FRC build season and competition season, we were able to use feedback from FRC teams to plan new features that we didn't conceive during the planning phase, and fix bugs that we hadn't discovered ourselves.

Milestone/Phase	**Start Date**	**End Date**
Background research and planning	March 2015	May 2015
Initial implementation	August 2015	December 2015
First public alpha release	November 2015	–
First stable release	January 2016	–
FRC build season	January 2016	February 2016

3.3 Architecture

GRIP is divided into two modules - the core and the graphical user interface (GUI). The core is responsible for performing the actual computer vision operations. The GUI allows the user to both manipulate the set of operations that the core performs and preview the outputs of the core.

Because of this separation, GRIP can run either in an interactive GUI mode or in a "headless" command line mode where the GUI is not present. Users can construct algorithms graphically and have the same behavior on an embedded processor with no modifications. This allows GRIP to function both as a tool for experimentation and as a practical production-ready solution for computer vision.

GRIP's core architecture is divided into three basic roles - image acquisition, the processing pipeline, and publishing. Figure 5 shows how data flows through the major components of GRIP and how they fit into the architecture of a larger robotic control system.

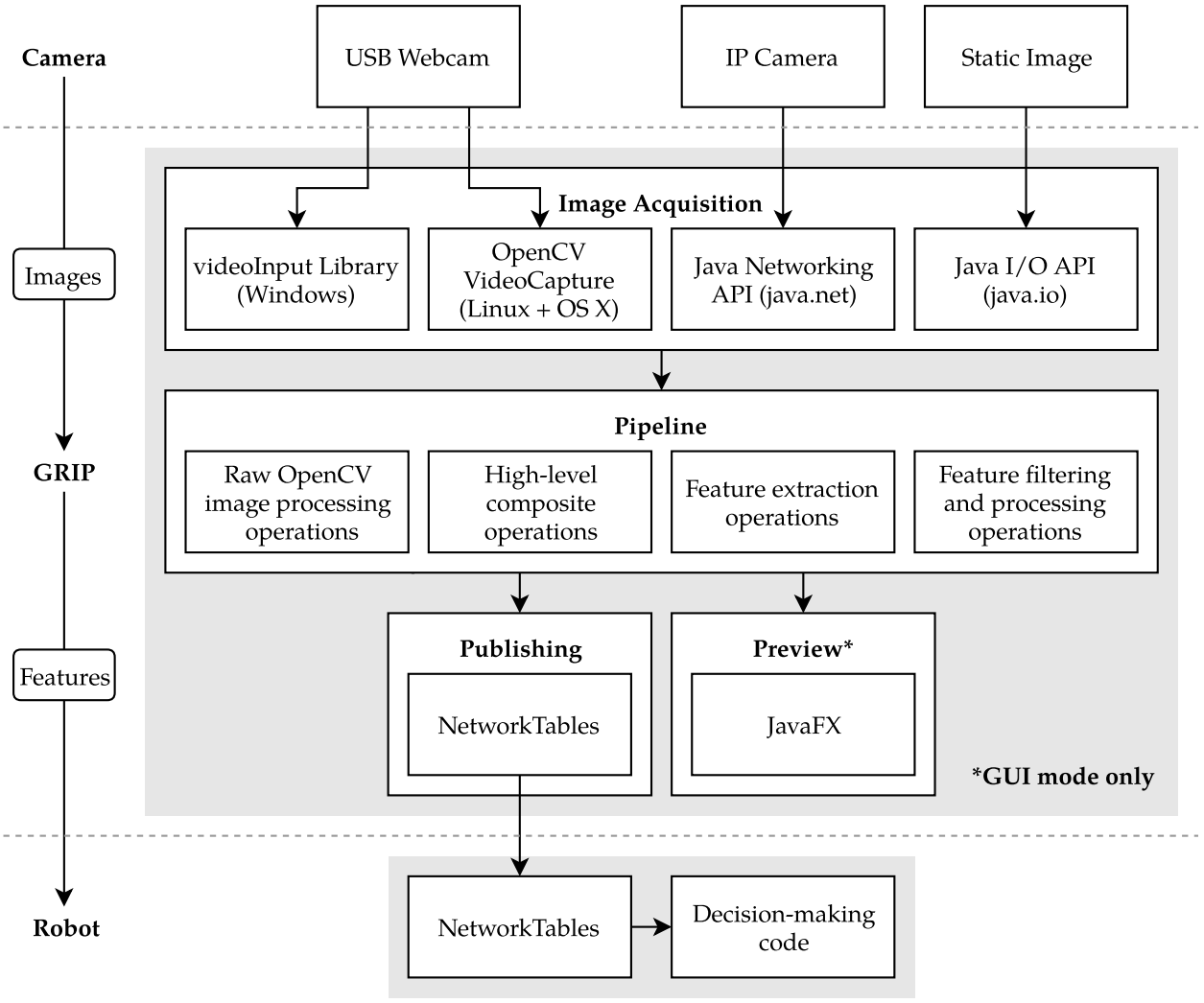


Figure 5: GRIP architecture

Several different input methods exist in GRIP for acquiring images, depending on the hardware and the operating system. In addition, a static image source can be used. The main use case for this is testing algorithms. Instead of developing an algorithm using a real-time camera feed that requires the physical presence of an object of interest and consistent lighting conditions, the static image source can be used with a list of existing sample images. For *FIRST* STRONGHOLD, over four hundred images of the field were provided to teams. The expected work-flow is that teams would create an algorithm in GRIP using many static image inputs as test data, then replace them with a physical camera input for actual use.

The processing pipeline performs the actual computer vision operations in GRIP. It consists of a series of steps that fall into four main categories.

- Raw OpenCV functions
- High-level composite operations, such as a generic "Blur" operation that can use several different filters, or a "Threshold" operation that segments an image based on HSV values
- Feature detection operations, which take an image as an input and produce a "report" of high-level features found in the image, such as contours or lines.
- Feature filtering and manipulation operations, which are used to narrow down features to ones that seem like the object of interest.

Finally, the ultimate output of the processing pipeline is sent to publishing operations. GRIP itself does not perform any decision making or robot control, it only produces data about features it finds, such as a list of coordinates, and publishes this data for another part of the robot control system to use.

One method we implemented for sending this data was NetworkTables, a network protocol used in FRC to create a shared key-value store between robots and other clients. [10] Using NetworkTables to communicate data to the control system means that GRIP could run on any machine — the roboRIO, the driver station, or a co-processor. If a team frequently runs into problems with GRIP running out of memory, they could simply run GRIP on a dedicated vision processing machine connected to the same network without modifying their own code.

NetworkTables is only used in FRC, so for GRIP to be useful for more general-purpose use cases, more publishing operations will have to be implemented. See also subsection 5.

Another method we implemented for sending data was ROS, a network protocol used by researchers and some robotic companies to relay data between different process. ROS uses a topic based publisher/subscriber infrastructure to allow different processes to interact. Using ROS to communicate data to the control system means that GRIP could run on any machine connected to the ROS master node.

3.4 User Interface

The distinguishing feature of GRIP is its graphical user interface, which is designed to allow users with a range of experience levels to author computer vision algorithms. Because computer vision applications are essentially pipelines of many small, commonly-used algorithms, GRIP's UI is based on the concept of a "step", which is a single primitive computer vision operation. Different steps can be connected together, allowing the output of one algorithm to be used as an input to another. From this basic building block, complicated computer vision pipelines can be constructed.

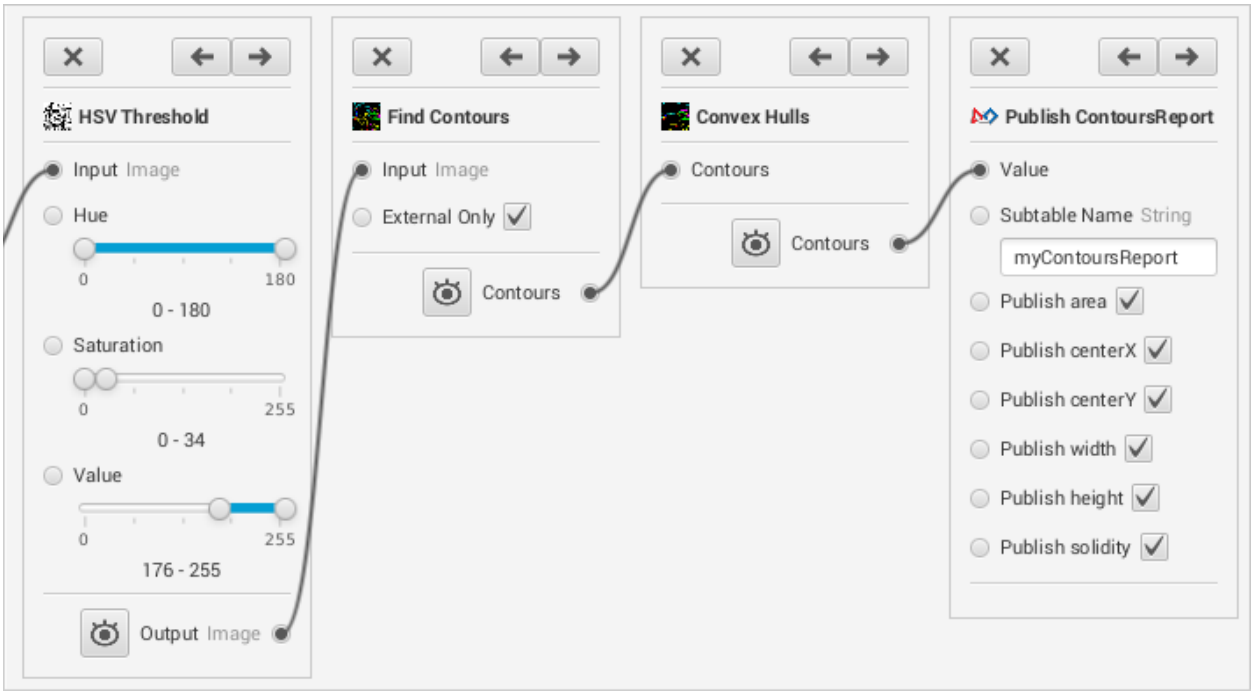
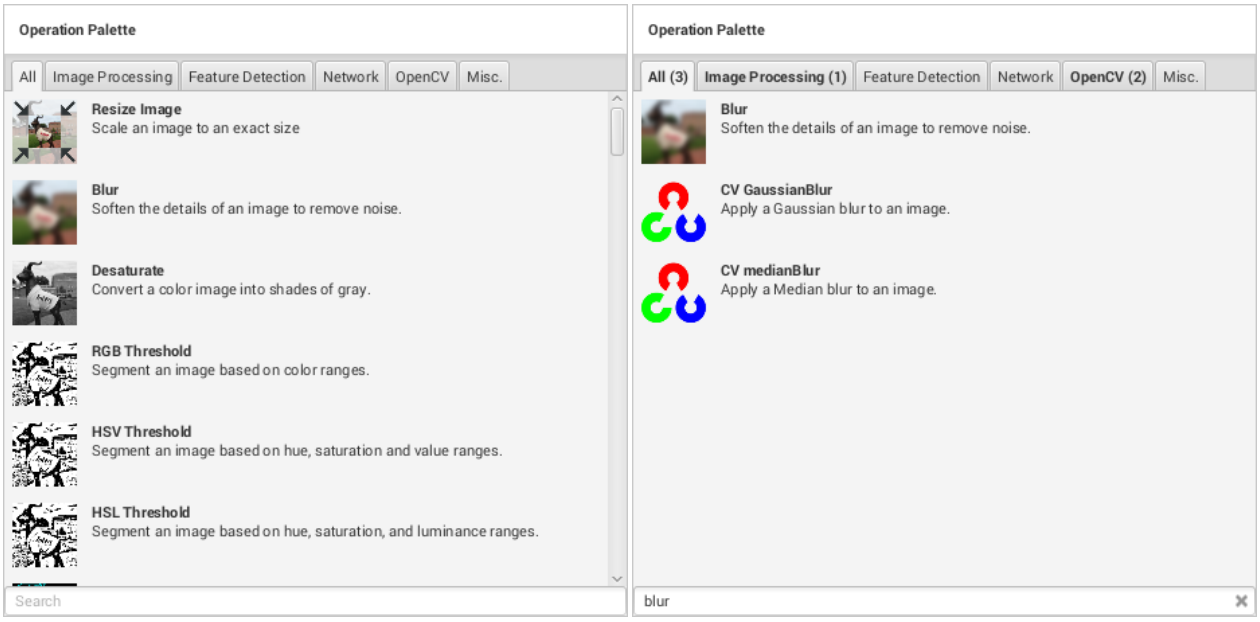


Figure 6: A simple GRIP pipeline

Figure 6 shows a simple pipeline with four steps: an HSV threshold to segment an image into black and white regions, a "find contours" step to extract contours representing the borders around the white regions, a "convex hulls" step to process these contours, and finally a publish step to write several measurements of the contours to NetworkTables. The labeled circles on the left and right of each step are the sockets for the inputs and outputs of that step respectively, and the curved lines between them indicate connections. Users can create new connections by clicking and dragging from one socket to another.

Some input socket values can be set with a UI field instead of by connecting it to another socket. For example, the threshold range for "Hue" in HSV Threshold is specified by the user with a ranged slider control. Output socket values have a toggle button that enables a preview of the value currently in that socket. The combination of these two features allows users to experiment with parameters in real-time and see the output, without the need to restart a program to test out a different combination of inputs.



(a) The normal palette appearance (b) The search feature

Figure 7: GRIP's operation palette

GRIP has over 60 operations available to use in the pipeline. However, novice users may not know the name of a specific operation, so the application includes a categorized, searchable palette, as shown in figure 7. This operation palette shows a name, description, and icon for every available algorithm that should help users find useful operations for their use cases. Clicking and dragging an operation adds it to the pipeline.

To the left of the pipeline is the sources panel, which contains inputs such as cameras and static images that serve as the starting point of an algorithm. Inputs, like steps, connect to the rest of the pipeline using sockets, providing a consistent paradigm for interacting with a GRIP algorithm.

Figure 8: The GRIP sources panel

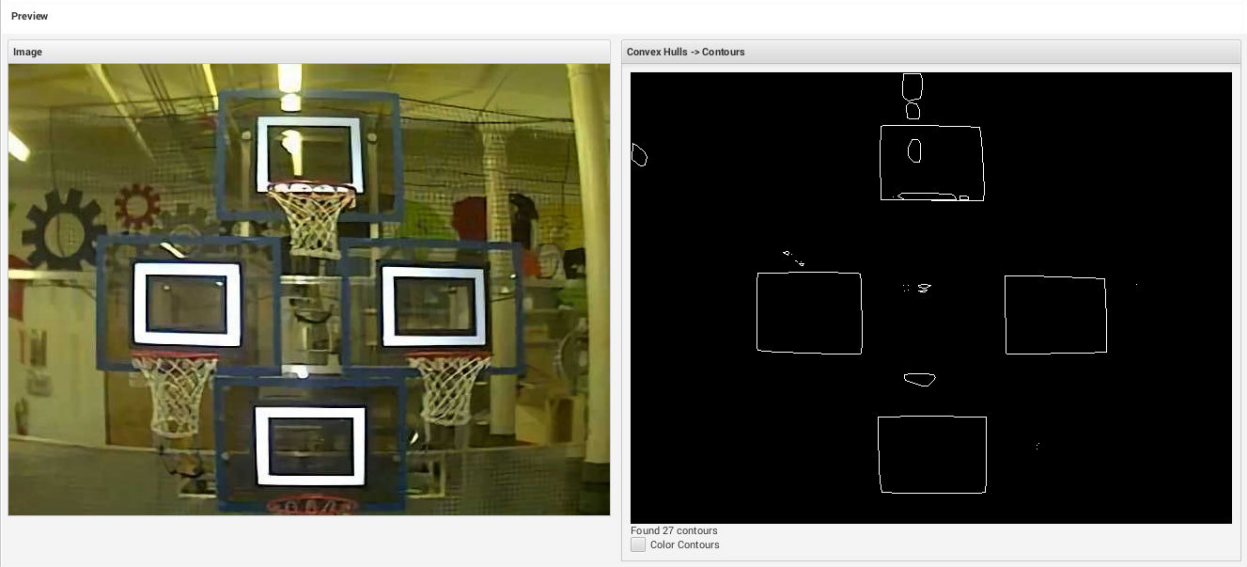


Figure 9: The GRIP preview panel

GRIP also contains a preview pane. Most sockets can be previewed in GRIP by toggling a button next to the socket name. Numbers and vectors are represented as text, images are represented by drawing the image, and high-level features like contours and lines are shown by rendering the shapes to a bitmap, optionally overlayed onto the input image. The values in these sockets show up in the preview pane and update in real-time, allowing users to gain instant feed back on modifications of an algorithm. For example,

a user may preview a contour list while adjusting the parameters of a "Filter Contours" operation to find the right minimum and maximum areas to use. Another use case that a computer vision novice may use is previewing the image resulting from an image processing operation, to visually learn what different operations do and how their parameters affect the output.

Figure 10: The GRIP deploy dialog

A dialog is included in GRIP to help users deploy algorithms to a dedicated machine, such as a robot controller. The dialog, shown in figure 10, presents a simple user interface that copies a pipeline to another machine using SSH (Secure Shell). GRIP does not need to already be installed on the machine, since all necessary code and data is included in the deploy — only a Java Runtime Environment is required to be installed. The deploy dialog by default contains values that should work for FRC teams deploying to a roboRIO without any modifications, but it could potentially be used by any other controller that supports SSH and Java.

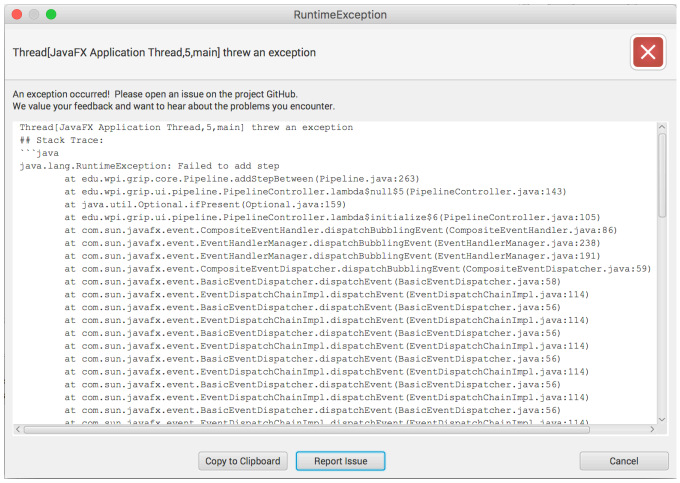


Figure 11: The Exception Alert Dialog

One of the most powerful debugging tools that GRIP has is a UI to handle unexpected exceptions while GRIP is running. The dialog, shown in figure 11, appears when GRIP runs into an uncaught exception. This dialog contains two buttons that allow a user to quickly open an issue on GitHub.

System Info:

Property Name	Property
GRIP Version	1.0.0
java.version	1.8.0_66
javafx.version	8.0.66
os.name	Windows 7
os.version	6.1
os.arch	amd64

Figure 12: Generated system information as rendered markdown

Additionally, the error message contains properly formatted markdown containing important details about the error including the full stack trace, and the users system information. This system information, seen in figure 12, is incredibly important for tracking down bugs that are system dependent. Fore example, during the development of GRIP we had a reoccurring bug that was only reproducible on Windows 7.

3.5 Implementation Details

3.5.1 Java and JavaFX

We wrote GRIP using the Java programming language. Java runs in a virtual machine, so it can be compiled once and run on multiple operating systems. Oracle describes the Java platform as intended to support "secure, portable, high-performance applications" [11]. Writing the programming in Java was a natural choice because one of our goals was to support many common operating systems and architectures, and because Java is a very common programming language, making it more likely that users could contribute modifications to GRIP.

JavaFX is a modern user interface toolkit for Java. We chose to use JavaFX for GRIP because it's included by default with the Java Runtime Environment, making it ubiquitous across desktop computers. JavaFX also has consistent appearance and behavior across different operating systems, which usually use different frameworks for user interfaces. This consistency allowed us to avoid duplicating our efforts in developing and testing the UI while still being able to commit to supporting multiple platforms.

3.5.2 OpenCV

We used OpenCV to perform the underlying image processing and computer vision operations in GRIP. OpenCV already supports a comprehensive set of performant operations, so adopting it saved us the trouble of developing our own. In addition, because OpenCV is already so widely used and well documented, using it as the underlying engine for GRIP makes it easier for experienced computer vision users to understand our source code and make modifications.

3.5.3 Build

The build system compiles, tests, builds and releases GRIP. Solid build infrastructure allows developers to add new features quickly and test them locally. This fast closed loop development cycle is critical to any software development project. Additionally, a solid

set of tests ensures that adding new features does not break existing functionality. Having solid build infrastructure is also critical in allowing new developers contribute to the project with minimal system configuration. For GRIP, we wanted to make it possible for a new developer to run a single command and have everything correctly configured and running. We were able to achieve this with Gradle. Gradle provides an entire ecosystem for managing a project's dependencies as well as compiling, running and testing a project. Currently, it is as simple as cloning the project and running one command to build GRIP.

3.5.4 ROS Build System

By using ROS Java we were able to include the ROS infrastructure in GRIP without creating a dependency upon the ROS environment. A build system was developed for ROS that allowed GRIP's custom messages to be generated within a Docker image agnostic of the developers build system. Additionally, our own custom maven repository was deployed onto GitHub to allow the java implementation of our messages to be acquired by our build system. The result was the ability to take advantage of ROS if it was installed and running on the host machine but it would not effect the user if the machine did not have ROS.

3.5.5 Generator

Many OpenCV methods conform to a very similar method signature structure, which several inputs and typically one output. Many operations we wanted to add would be very repetitive to write manually. As many of these operations were all very similar to each other in syntax structure we determined that a code generator could be used to parse the OpenCV source code, find the operations we wanted to include in GRIP, and then automatically generate a piece of source code to add that operation in GRIP.

The GRIP generator was implemented using JavaParser, an open source project for parsing Java source code into an abstract syntax tree. We were able to use this abstract syntax tree to automatically find methods in OpenCV and generate operations based on their name and arguments. Over thirty operation in GRIP are generated using this technique.

As a result of the generator, a large subset of the operations available directly

to OpenCV users are also available in GRIP. This expands the amount of functionality offered by the program, and also makes it easy for seasoned OpenCV users to migrate their development to GRIP.

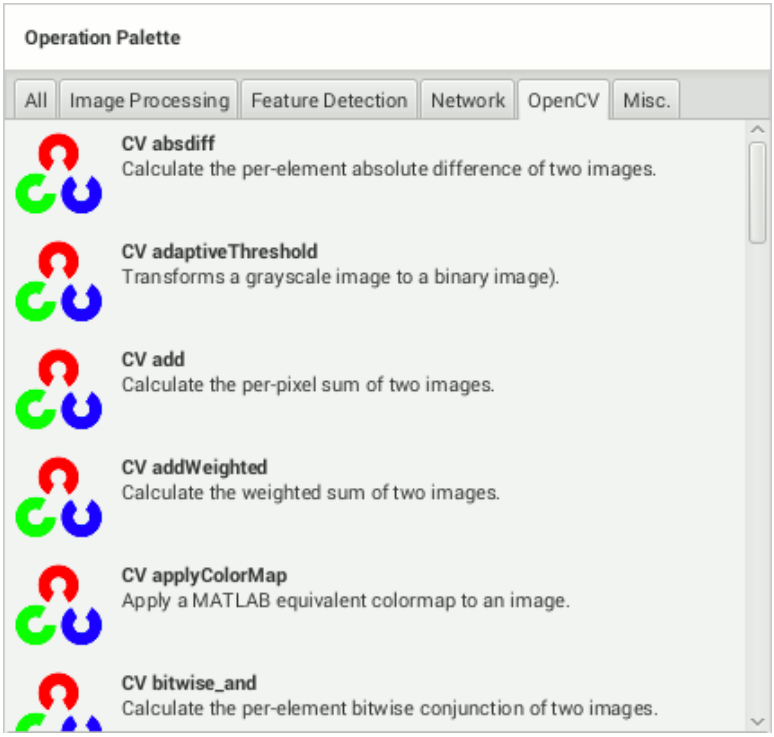


Figure 13: Some of the generated OpenCV operations in the GRIP user interface

3.5.6 Project Layout

Gradle defines a very specific project layout that it recommends in order to allow a project to compile without explicit configuration. Originally, the project was confined to a single module that included both the Core code and the JavaFX UI code. Eventually, a build step was required to add the generator.

Gradle defines a directory `buildsrc` that can contain any project specific plugins that your project uses. This is where the generator code was added so that it would run as a pre-compile step for every build of GRIP. Later, when working on a methodology to deploy GRIP to a remote device to run in headless mode we found that it would be much easier to deploy and run a single JAR. In order to produce a JAR that contained

all of the dependencies for the GRIP core source we needed to split GRIP into two separate modules, the core that would produce a JAR with all of its dependencies inside that would be depended upon by the UI module to run. This allowed us to have one jar that we could SCP over to a remote device and run without trying to keep track of the core's dependencies within GRIP.

3.5.7 Tests

Testing is a fundamental part of the software development process. It ensures that new code being added works as expected and that as the code evolves exiting functionality isn't broken. There are two fundamental testing methodologies: Unit, and Integration. Unit testing is designed to test the smallest unit of code to ensure that it functions as expected. Integration testing, by contrast, tests that when all of the source code is integrated together the data flows through the entire program in an expected way. The GRIP tests are a mix of both Unit and Integration tests.

JUnit is described as a simple, open source framework to write and run repeatable tests [12]. It provides assertion based testing of testing classes in expected ways. For example, it allows you to assert that a value is true, false, null, equal to something, ect. When assertions fail they provide valuable feedback regarding the cause of the problem to allow a developer to quickly debug the problem and resolve it. By developing a solid suite of unit tests during the development of GRIP we have ensured that future refactors do not introduce unintended bugs and expected functionality is preserved.

TestFX is self described as "Simple and clean testing for JavaFX" [13]. It provides an API to allow a developer to quickly automate UI tests for JavaFX. Using it's Fluent API you can easily tell the automated test system to click on a button with the text "OK" and validate that the UI and data model reacts as expected. Instead of assertions TextFX allows you to `verifyThat` specific UI elements are present or not. Additionally, TestFX depends upon the JUnit test life cycle allowing you to take advantage of JUnit assertions while testing. GRIP uses TestFX to ensure that the various components that make up the UI render correctly and fully as well as to exercise the various user inputs.

3.5.8 GitHub Webhooks & Services

Github has a directory of integrations that can be easily configured to customize how the repository behaves. They allow you to automate your repository to perform specific actions when a pull request is opened, when a release tag is pushed, when code is merged into a repository. Our project uses several of these webhooks to help automate the management of the project and keep us informed when things change. Additionally, GRIP's build infrastructure is resilient against developer mistakes and buggy code. The GRIP repository takes advantage of GitHub's "Protected branches" to prevent developers from being able to push code to master without first passing configured PR status checks. These checks significantly decrease the likelihood that a developer will break the build for others. It was an expectation that GRIP will continue to be developed for many years beyond the scope of this MQP. We wanted to ensure that it could maintain itself and evolve safely beyond the original authors involvement.

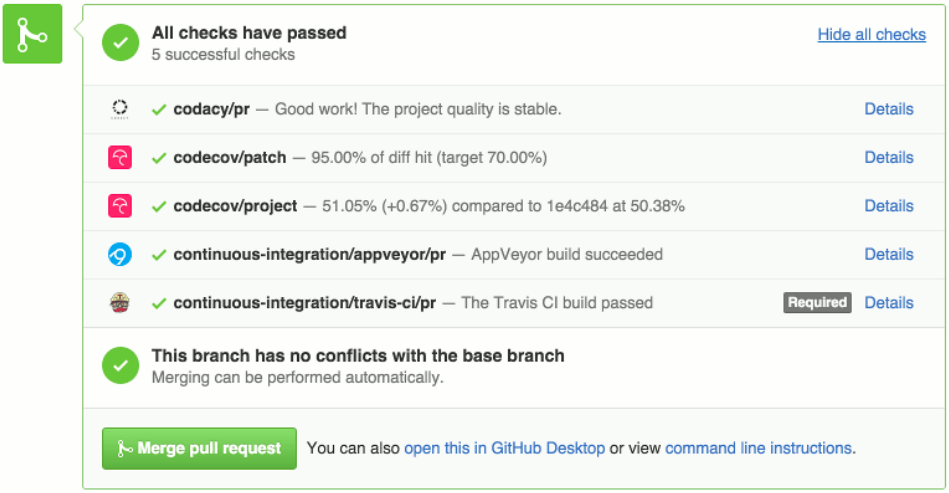


Figure 14: GRIP Pull Request WebHooks Status Checks

3.5.9 Gitter

Gitter is similar to IRC chat for projects but with integration with GitHub and many other services that offer WebHooks. This allows you to very easily reference issues or pull requests and Gitter will automatically create a link directly to that issue. Additionally, all

status changes for the repository are reflected in the status bar on the right side of the chat. This allows us to know quickly when a Pull Request succeeds or fails and if a Pull Request that has just been merged breaks master. Gitter also includes an integration with Trello which we were originally using to manage our projects burn-down board.

We have utilized Gitter to communicate with the growing number of outside developers who have become involved with GRIP. It allows us to ask them questions about how they are using GRIP, provide support when they are facing an issue, talk about future development plans, share code samples, and encourage them to open pull requests with features they want. Through Gitter we have been able to watch as a community has formed around GRIP with different users supporting one another with the technical challenges that computer vision poses. The closed loop feedback loop with users has been instrumental in the success of GRIP.

3.5.10 Codacy

Codacy is a relatively new service that provides automated code reviews on pull requests using PMD and other similar static code analysis tools. When it encounters one of the code problems you have set it to check for it automatically leaves a comment on the pull request. This includes things varying from unused imports to forgetting to calling `new Exception()` with out a `throw` in front of it. It then adds its own status check to the PR (pass or fail).

Codacy catches many things that may go unnoticed during a normal code review. It provides defense against many common but obscure coding mistakes. This is important as an open source project becomes popular. Reviewing many pull requests can be time consuming. Codacy provides automated feedback that can be resolved before a real person has the opportunity to provide a review.

3.5.11 Continuous Integration

Travis CI (Travis) and AppVeyor are a continuous integration service provided for free to open source projects. Much like Jenkins you can setup these services to run code on their servers and report back whether or not specific checks pass or fail. Additionally, you

can set up these services to perform specific actions on release. Travis CI provides Linux and OSX operating systems while AppVeyor provides a Windows Server. As AppVeyor is significantly slower than Travis we only require the Travis check to pass currently before a PR can be merged. Every time that anyone working on GRIP creates a pull request against the master repository Travis CI and AppVeyor grabs the code from the branch and runs specific commands they read (each respectively) from the `.travis.yml` and `appveyor.yml` files in the root directory of the repository.

AppVeyor AppVeyor is our Windows continuous integration service. Although it is much slower than Travis CI it serves the vital role of building our 64 bit Windows release. As the core developers on this project don't have easy access to a Windows OS in order to build releases AppVeyor will automatically build any release tag that it is given. Additionally, it runs the Gradle check task to ensure that all of the tests also pass on Windows. There have been circumstances where UI components that have appeared correctly on Linux and OSX have not worked correctly on Windows. AppVeyor also sends a notification webhook to the Gitter chat whenever it passes or fails to provide feedback on build failure/success.

Travis CI Travis CI is our primary workhorse when it comes to our continuous integration system. Once Travis receives a PR it runs the core and UI tests compiling code coverage metrics with JaCoCo. Once the tests complete successfully a small python script from CodeCov retrieves the JaCoCo code coverage and sends it off to their servers to be processed. As part of the build we also generate the project's JavaDocs. When a PR is merged with master the Javadocs are automatically pushed to the repo's gh-pages branch. This allows anyone to view the documentation for the project online without having to build the project on their machine. After the build is complete Travis sends a notification to Gitter though a webhook informing us of whether or not a build has passed or failed.

3.5.12 Packaging and Deployment

The Java development tools can create operating system-specific installers for applications developed with JavaFX. The installer bundles all dependencies with the application, so

users are not required to have a particular Java Runtime Environment already installed.

GRIP builds for Windows and Ubuntu are generated on Travis and AppVeyor automatically with each release. A developer must explicitly generate builds for other platforms, including other Linux distributions and OS X, by running a single command.

4 Usage and Results

GRIP has close to 7,000 downloads from GitHub [14]. The project has over 100 stars on GitHub.

4.1 Usage by FRC Teams

FIRST reports teams using computer vision in their robots has significantly increased due to the availability of GRIP. During a WPI *FIRST* Robotics Competition held after kickoff, several teams were queried regarding their initial experience with GRIP. Of the teams that chose to use GRIP to perform their computer vision processing, most teams found the UI to be intuitive.

4.2 Open Source Community

Even before the first stable release of GRIP, members and mentors from *FIRST* Robotics Competition teams became interested in GRIP. Several developers opened pull requests to add features they found useful. Once GRIP had its first full release after the *FIRST* kickoff. we were overwhelmed with people opening issues discussing feature requests and defects. In many cases, this helped to drive the development cycle. Currently there are around half a dozen individuals who actively provide support to new people trying out GRIP to solve computer vision challenges.

4.3 Researchers

Junior Cunha, a Former WPI Robotic Student currently teaching and researching Robot Perception in Rhode Island provided this quote.

> Robot Perception is an intricate part in developing pragmatic and interactive-centric robotic systems with the objective of solving real-world problems. A plethora of robot perception capabilities are derived from understanding and

> extracting rich semantics from various different image modules. Thus, innovative and efficient methods for image processing have become symbiotic with to development of smart and complex systems. OpenCV, a computer vision software, is an ideal platform for researchers for it is written in general-purpose languages, C++ and Python, it is composed of intuitive modular software structures, and it allows for state-of-the-art algorithms to be implemented with ease. Nevertheless, OpenCV libraries and functions are not bug-free, and implementing even a simple function can be time-consuming and require the user to possess extensive knowledge and experience with CV systems. GRIP is an alternative solution to such burdens, for it allows fast-prototyping and juxtaposing algorithms and image pipelines through a drag-and-drop GUI. Such platform reduces vision pipeline assembly- time significantly and allows researchers to quickly test pet theories and modify function settings. Modularity, portability, and feasibility accentuates GRIP utility for solving complex image processing challenges in a robust and precise manner.

4.4 Corporate Usage

The response from corporations has been surprisingly positive. Two companies have discussed the possibility of using GRIP to try to solve their robotic computer vision challenges.

Additionally, another company that is actively involved with the development of GRIP provided this quote:

> Artaic is a company in Boston that uses robots to create custom mosaics. We're building a new robot that will use computer vision to locate tiles. With hundreds of kinds of tiles, the vision module needs to be able to rapidly switch between pipelines multiple times per second, and if a new kind of tile is added, an operator will need to be able to quickly and easily create a new pipeline to locate that tile without having to write any code.
>
> We looked at industrial software packages for graphically creating computer vision pipelines, but none were able to switch pipelines. So we turned to GRIP,

which, because it is open-source, we were able to modify to meet our needs [15].

5 Conclusion and Future Work

This project's goal was to create a solution for both inexperienced and seasoned computer vision users to rapidly create computer vision algorithms. After researching CV theory and the state-of-the-art in CV tools, we designed GRIP to fulfill this goal. With the combined effort of our project team and contributions from interested participants of *FIRST* Robotics Competition, we created a complete, usable graphical interface for developing, testing, and deploying computer vision algorithms.

GRIP was very successful in completing our goals. Many FRC teams chose to use GRIP in the 2016 *FIRST* Robotics Competition game, increasing the exposure to computer vision that many beginning programmers receive. This demonstrates GRIP's ability be used by inexperienced users for simple tasks with severe constraints on development time. In addition, Artaic, a company that offers robotic custom tile services, has invested resources into adopting GRIP for commercial use, showing that that the program is also extendable enough to provide value for more advanced use cases. Due to GRIP's interacting with an open source community, further development of the program is still taking place after the completion of this project.

5.1 Recommendations for Future Work

There are several missing or in-progress features in GRIP that would further extend the number of use cases it covers.

Network Protocols Support for a wide variety network protocols besides NetworkTables would create an opportunity for GRIP to be used in more existing systems and therefore see higher adoption. For example, as of this report, support for Robot Operating System (ROS) and HTTP have been or are currently being written.

Code Generation GRIP currently runs computer vision algorithms itself and communicates with other programs though a network connection. Adding the ability to instead generate source code from a pipeline would allow GRIP to be used to develop algorithms

that run on a machine not powerful enough to run GRIP itself, or for projects where it's inconvenient to introduce a new network protocol.

Scripting One planned feature that was never fully implemented was the ability for users to write custom operations for GRIP in a scripting language. This would allow users to extend the program without learning about its internal structure or waiting for feature requests to be fulfilled.

More Platforms Although GRIP runs on most common operating systems and CPU architectures, we never created a version that worked on the Raspberry Pi, a small, inexpensive computer popular for hobbyist robotics projects. Support for the Raspberry Pi could make GRIP a go-to solution for makers working on small projects not worth the effort of learning a complicated CV toollkit like OpenCV.

Appendices

A Sample Code for GRIP in FRC

```
import edu.wpi.first.wpilibj.IterativeRobot;
import edu.wpi.first.wpilibj.networktables.NetworkTable;
import java.io.IOException;

public class Robot extends IterativeRobot {

    private final NetworkTable grip = NetworkTable.getTable("grip");

    @Override
    public void robotInit() {
        /* Start GRIP in a new process */
        try {
            new ProcessBuilder("/home/lvuser/grip &").inheritIO().start();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    @Override
    public void autonomousPeriodic() {
        /* Get published values from GRIP using NetworkTables */
        for (double area : grip.getNumberArray("targets/area", new double[0])) {
            System.out.println("Got contour with area=" + area);
        }
    }
}
```

B GRIP Source Code

The source code for GRIP is available on GitHub:
`https://github.com/WPIRoboticsProjects/grip`

6 Bibliography

[1] About OpenCV. Accessed 2016-2-18. [Online]. Available: http://opencv.org/about.html

[2] RoboRealm Robotic Machine Vision Software. Accessed 2016-2-16. [Online]. Available: http://www.roborealm.com/

[3] R. Klette, *Concise Computer Vision: An Introduction into Theory and Algorithms*. Springer, 2014.

[4] E. R. Davies, *Machine Vision: Theory, Algorithms, Practicalities*. Morgan Kaufmann, 2005.

[5] NI Vision Assistant Help. [Computer software].

[6] ROS About. [Online]. Available: http://www.ros.org/about-ros/

[7] What is FIRST Robotics Competition? FIRST. [Online]. Available: http://www.firstinspires.org/robotics/frc/what-is-first-robotics-competition

[8] Recycle Rush Game Description. FIRST. [Online]. Available: http://www.firstinspires.org/sites/default/files/uploads/resource_library/frc/game-and-season-info/archive/2015/2015-frc-game-description-1page.pdf

[9] (2016, February) 2016 FIRST Robotics Game Manual: FIRST STRONGHOLD. FIRST. [Online]. Available: https://firstfrc.blob.core.windows.net/frc2016manuals/GameManual/FRC-2016-game-manual.pdf

[10] (2015, June) Network Tables Protocol Specification, Version 3.0. Accessed 2016-2-20. [Online]. Available: https://docs.google.com/document/d/1nxiriSz1n_O4d7qWrzt16uaFjrU48mB-UvPE2AG0zmU

[11] Oracle. Java Technologies. Accessed 2016-4-27. [Online]. Available: https://www.oracle.com/java/technologies/index.html

[12] JUnit FAQ. [Online]. Available: https://github.com/junit-team/junit4/wiki/FAQ

[13] Test FX ReadMe. [Online]. Available: https://github.com/TestFX/TestFX/blob/master/README.md

[14] Github Release Stats. [Online]. Available: http://www.somsubhra.com/github-release-stats/?username=WPIRoboticsProjects&repository=GRIP

[15] S. Carlberg, on behalf of Artaic.