

Summer 8-17-2020

VisualOCV: Refined Dataflow Programming Interface for OpenCV

John Boggess
johnboggess@southern.edu

Follow this and additional works at: https://knowledge.e.southern.edu/mscs_reports



Part of the [Graphics and Human Computer Interfaces Commons](#), and the [Programming Languages and Compilers Commons](#)

Recommended Citation

Boggess, John, "VisualOCV: Refined Dataflow Programming Interface for OpenCV" (2020). *MS in Computer Science Project Reports*. 5.
https://knowledge.e.southern.edu/mscs_reports/5

This Article is brought to you for free and open access by the School of Computing at KnowledgeExchange@Southern. It has been accepted for inclusion in MS in Computer Science Project Reports by an authorized administrator of KnowledgeExchange@Southern. For more information, please contact jspears@southern.edu.

VISUALOCV: REFINED DATAFLOW PROGRAMMING INTERFACE FOR
OPENCV

by

John R. W. Boggess

A PROJECT DEFENSE

Presented to the Faculty of

The School of Computing at the Southern Adventist University

In Partial Fulfilment of Requirements

For the Degree of Master of Science

Major: Computer Science

Under the Supervision of Professor Hall

Collegedale, Tennessee

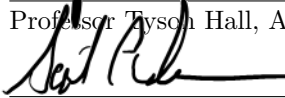
August 4, 2020

VISUALOVC: REFINED DATAFLOW
PROGRAMMING INTERFACE FOR OPENCV

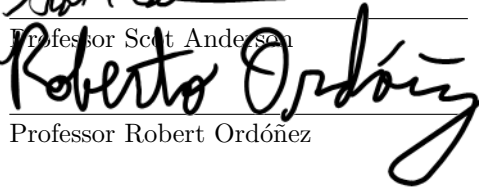
Approved by:



Professor Tyson Hall, Adviser



Professor Scott Anderson



Professor Robert Ordóñez

Date Approved 2020-08-04

VISUALOCV: REFINED DATAFLOW PROGRAMMING INTERFACE FOR OPENCV

John R. W. Boggess

Southern Adventist University, 2020

Adviser: Tyson Hall, Ph.D.

OpenCV is a popular tool for developing computer vision algorithms; however, prototyping OpenCV-based algorithms is a time consuming and iterative process. VisualOCV is an open source tool to help users better understand and create computer vision algorithms. A user can see how data is processed at each step in their algorithm, and the results of any changes to the algorithm will be displayed to the user immediately. This can allow the user to easily experiment with various computer vision methods and their parameters. EyeCalc 1.0 uses the Microsoft Foundation Class Library, an old GUI framework by Microsoft, and contains various bugs. This project recreated EyeCalc 1.0 with C# to create a back-end library and Windows Presentation Foundation for the GUI. The back-end library is written with .Net Standard 2.0, which will allow it to be easily ported to other platforms in the future. Various new features have been added, such as the ability to combine a set of operators into a macro, or the ability to add new operators without needing access to the source code.

Contents

Contents	v
List of Figures	ix
List of Tables	xi
1 Introduction	1
2 Background	5
2.1 Open Source Computer Vision Library	5
2.2 Competitive Analysis	5
2.3 Cross-platform vs. Native GUI Development	8
2.4 Usability	10
3 Implementation	13
3.1 Summary of Requirements	13
3.2 Development Approach	14
3.3 Task Delineation	15
3.3.1 Back-end	15
3.3.1.1 Creating New Operators	17
3.3.1.2 Managing Types	18

3.3.1.3	Operator Validators	20
3.3.1.4	Operator Communication	21
3.3.1.5	Marcos	22
3.3.1.6	Synchronization	22
3.3.2	GUI	22
3.3.3	Adding Operators	24
3.3.4	Operator Constructor	25
3.4	Usability Survey	26
3.5	Final Deliverables	26
4	Testing & Evaluation	27
4.1	Acceptance	27
4.2	System	27
4.2.1	Usability	28
4.2.2	Performance	29
4.3	Integration	30
4.4	Module	31
4.5	Unit	33
5	Conclusion	35
5.0.1	Future Work	35
A	Requirements Specification	37
A.1	Application Requirements	37
A.1.1	Operator Requirements	37
A.1.2	Connections	40
A.1.3	Macros	40

A.1.4	Workspace	41
A.1.5	General	42
B	Usability Survey	43
C	Acceptance Survey	49
	Bibliography	53

List of Figures

1.1	EyeCalc 1.0.	2
3.1	This shows what the operator XML files look like. The <Dlls> tag informs VisualOCV what DLLs contain the methods the operators represent. After this tag each operator is defined by identifying the type containing the method and the method the operator represents. The operator's name, inputs, and outputs follow.	18
3.2	This shows how to define multiple possible DLLs to reference an integer from, and save and load methods for an integer. Note that it has two attributes for AssemblyQualifiedName, one references System.Private.CoreLib for .Net Core and the other references mscorlib for .Net Framework.	19
3.3	An example of what defining validators looks like. The file is very similar to an operator XML file. The type containing the method must be specified and the DLL containing the type must also be specified. . .	20
3.4	This shows how a validator can be assigned to an operator's input in the operator XML file.	20

- 3.5 This diagram indicates how each operator communicates with each other. When an operator is done processing its data, it will send its output to each operator that output is connected to. The receiving operator will store that data, then wake up its processing thread to process the data. After the data has been processed, the operator will send its outputs to its connected operators. 21
- 3.6 This shows what the application currently looks like in its light theme (a) and dark theme (b). The top contains the menu and toolbars. The left side of the screen holds the list of operators. In the middle of the screen is the main workspace. Inside the main workspace is the operators. 23

List of Tables

2.1	OpenCV Functionality [1]	6
2.2	Cross-platform GUI Frameworks	9
3.1	Sprints	16
3.2	Supported C# Operators	25
4.1	System Tests	28
4.2	Usability Test Results	28
4.3	Integration Tests	30
4.4	Module Tests	32

Chapter 1

Introduction

Developing computer vision algorithms can be an iterative and time-consuming process. Algorithms can be difficult to create and lots of tweaking may be required to get desired results. The EyeCalc desktop application was originally developed to provide the user with a dataflow programming environment for developing computer vision algorithms. Users can create and modify algorithms and see the results of their changes in real time. The application provides the user with operators, each of which represent an OpenCV function or object. Each operator has a set of controls that represent parameters of the function or object. Operators can be connected so that the output of one operator is passed to other operators for processing. Whenever an operator's inputs or controls are modified, it processes the new data and shows the results to the user.

EyeCalc was originally funded by the Center for Innovation and Research in Computing (CIRC) and was created by Hai Vo and Micheal Dant for the School of Computing at Southern Adventist University. It was written in C++ and used the Microsoft Foundation Class (MFC) Library for the GUI. EyeCalc 1.0 is a subsequent open source revision that included bug fixes, an updated UI, and more operators

[3]. A sample of EyeCalc 1.0 can be seen in Figure 1.1. A large number of operators is necessary for EyeCalc to be a useful application; however in its current state it only has 83 operators and more can not be added without modifying the source code. EyeCalc 1.0 also contains various bugs, including the following:

- flickering UI
- UI elements being drawn in the wrong order
- small memory leaks in certain situations
- loading an algorithm may not restore connections between operators
- crashes may occur on invalid operator input
- parallel paths that converge are not synchronized

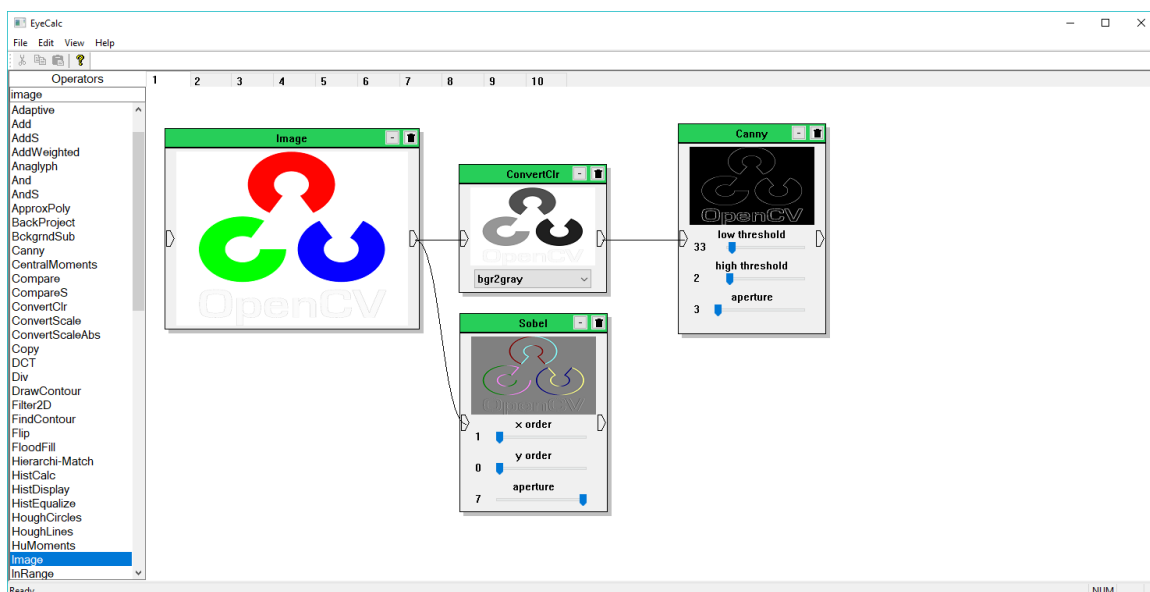


Figure 1.1: EyeCalc 1.0.

This paper reports on the creation of VisualOCV, which is a C# translation that uses Windows Presentation Foundation (WPF) for its GUI, adds new features,

and more operators. The main logic of the application has been developed with .Net Standard 2.0 to make it possible to run on multiple platforms; however, development has been focused on Windows 10.

Chapter 2

Background

2.1 Open Source Computer Vision Library

Open Source Computer Vision Library (OpenCV) is an open source library created by Intel for real-time computer vision and machine learning. OpenCV is very popular, with an estimated 18+ million downloads, and is used by several large companies such as Intel, Google, and Microsoft [2]. It officially supports interfaces for C++, Python, and Matlab and runs on Windows, Linux, Mac OS, iOS, and Android. OpenCV contains more than 2500 optimized algorithms [2]. A few of these include facial recognition, background subtraction, object identification, object pose estimation, motion tracking, and image stitching. As shown in Table 2.1, OpenCV is organized into fifteen main modules. [1].

2.2 Competitive Analysis

There are several development applications that allow users to utilize the OpenCV library. MATLAB, a popular application used for data analysis, creation and

Table 2.1: OpenCV Functionality [1]

Module	Functions
Core Functionality	Definition of basic structures
Image Processing	Linear and nonlinear filtering, geometric transformations, color space conversion, and histograms
Image File Reading and Writing	Reading and writing images
Video I/O	Video capturing and video codecs
High-level GUI	Simple UI capabilities
Video Analysis	Motion estimation, background subtraction, and object tracking
Camera Calibration and 3D Reconstruction	Single and stereo camera calibration, object pose estimation, and 3D reconstruction
2D Features Framework	Salient feature detectors, descriptors, and descriptor matchers
Object Detection	Object detection
Deep Neural Network Module	Framework for deep learning
Machine Learning	Statistical classification, regression, and clustering of data
Clustering and Search in Multi-Dimensional Spaces	Utilizes FLANN for fast nearest neighbor search in large datasets and for high dimensional features.
Computational Photography	Inpainting, Denoising, HDR imaging
Images Stitching	Join multiple images into one image
Graph API	-Framework to make image processing fast and portable

simulation of mathematical models, has an OpenCV interface to allow users to use OpenCV functions [4]. The MATLAB language is simple to use, which makes it easier to write computer vision algorithms with OpenCV. However, there is no visual editor for writing OpenCV algorithms. MATLAB is also not free and the interface requires the user to purchase two additional toolboxes.

MATLAB does allow users to create their own toolboxes. One third-party toolbox utilizing OpenCV is Peter Corke's Machine Vision Toolbox [5]. It offers over 100 functions to be used for image and video processing.

Another application that allows users to utilize the OpenCV library is Laboratory Virtual Instrument Engineering Workbench (LabVIEW). LabVIEW offers a graphical programming approach to system design, data acquisition, signal processing, and test automation. LabVIEW has an official package for OpenCV, but it requires the user to purchase more software in addition to paying for LabVIEW [6]. However, with some work, users are able to call OpenCV functions directly.

The OpenCV website references a couple of visual tools to assist with the creation of algorithms. The first of these is the OpenCV Demonstrator [7]. This tool allows users to experiment with some basic functions without having to write any code. The tool provides several dozen algorithms with modifiable parameters that immediately show the results of changes. However, there is no way for the user to combine these algorithms or add new algorithms. OpenCV also comes with a visual debugger [8]. This debugger can allow users to see how an algorithm processes an image at each step. However, there is no real-time component to it. The user must stop the algorithm, make changes, recompile, and run the program to see what effect changes have on the image.

BlendOcv combines Blender and OpenCV to create a dataflow programming environment for OpenCV [9]. Users can connect blocks together, modify their

parameters and see results in real time. The application also claims it can run on multiple OSs, though it does not state which ones. However, there is a small number of OpenCV functions implemented and the project has not been updated since 2014.

2.3 Cross-platform vs. Native GUI Development

When developing an application that can be run on multiple platforms, the developers must decide if cross-platform GUI framework will be used or if native GUI frameworks will be used for each platform. Using a cross-platform framework can significantly reduce the time needed to create GUI for multiple platforms since only one GUI needs to be created instead of one for each platform. It also makes the project easier to maintain and update since only one code base would need to be updated [18]. However, cross-platform frameworks can suffer from various issues. Cross-platform GUIs may sacrifice performance to be able to run on all supported platforms. These GUIs also lose functionality since they are not able to take advantage of platform-specific features [19]. Native GUI frameworks are able to take better advantage of platform resources and features, increasing performance and functionality. An application with a GUI that was created natively will have a native look and feel while a cross-platform GUI may not, making the application look alien or feel awkward to use. However, using native frameworks requires a GUI to be created for each supported platform. This causes lots of code duplication, which can make the project difficult to maintain and update across each platform [20].

There are several cross-platform GUIs available for C#. Some information is given on these frameworks in Table 2.2. Qml.Net, Xwt, and QtSharp do not have

Table 2.2: Cross-platform GUI Frameworks

Framework	Documentation	Community Adoption	Development Stage	Native Look & Feel
Avalonia	●●●○	●●○○	Beta	Yes
Gtk#	●●●●	●●●●	Release	No
Qml.Net	●○○○	●○○○	WIP	Yes
Xwt	●○○○	●○○○	Release	Yes
QtSharp	●○○○	●○○○	Alpha	Yes
Eto	●●●○	●●●○	Release	Yes

Key: ●○○○ - Poor ●●○○ - Fair ●●●○ - Good ●●●● - Excellent

much documentation and seem to have little community adoption [10, 11, 12]. Avalonia [13] has good documentation and has some decent community adoption. It is also extremely similar to WPF, which is extensively documented and provides a lot of functionality. However, it is in beta and thus suffers from typical beta issues, and may have updates that break projects in the future. Experimenting with Avalonia reveals some issues that makes laying out the UI a bit difficult at times. Gtk# and Eto both are well documented and have good community adoption [14, 15]. Gtk# is a large project that provides a lot of functionality but it only easily provides a native look and feel for Linux. Eto provides native look and feel for its supported platforms, but is a bit lacking in functionality.

C# has official GUI frameworks for native Windows GUIs such as Windows Forms and Windows Presentation Foundation (WPF) [16, 17]. Windows Forms was a GUI framework included in the .Net Framework by Microsoft. Windows Forms is extensively documented and has received a large amount of community support. WPF was released by Microsoft after Windows Forms and is also well documented and has a large amount of community support.

2.4 Usability

A user's experience (UX) with a product ultimately determines their perception of the product and the product's success. One way of quantifying the UX of an application is to find how usable it is. There are various ways to measure an application's usability. One common and easy approach is to gather a small group of people and give them a set of tasks to perform with the application. Afterwards, the participants will fill out a short questionnaire. The scores given by the participants can be used to determine the usability of the application. Some common questionnaires include the System Usability Scale (SUS), Questionnaire for User Interaction Satisfaction (QUIS), and the Computer System Usability Questionnaire (CSUQ). The SUS is the most common usability questionnaire, making up about 43% of post test questionnaires [21]. The SUS consists of ten five-point Likert-scale questions. Each scale ranges from strongly disagree (1) to strongly agree (5). The values of the odd questions are subtracted by one, and the values of the even questions are subtracted from five. After this, the scores are added up and multiplied by 2.5 to make a 0-100 range. The SUS has become popular due to its shortness, ease of scoring, and reliability [21]. However, it can be somewhat difficult to determine exactly what each SUS score means and what constitutes a good score. Analysis of many SUS questionnaires shows that a score of approximately 70 is acceptable [22]. A possible improvement to the SUS is to include an eleventh question about the overall user-friendliness of the product [22]. It is a seven-point scale with an adjective rating, ranging from 'worst imaginable' to 'best imaginable'. Testing this adjective rating scale shows that it is highly correlated with the score calculated by the other ten questions [22]. This added question could help practitioners determine more precisely what their SUS scores

mean.

The Usability Metric for User Experience (UMUX) and UMUX-LITE are two new questionnaires that claim to be highly correlated with the SUS, and thus, also highly reliable [21]. These two questionnaires consist of seven point Likert scales, and have less questions than the SUS. The UMUX has four questions and the UMUX-LITE has only two. The high reliability and shortness of these questionnaires could make them useful in usability tests where participants are asked many questions. However, there has been some debate about the reliability of the UMUX, as other researchers have not always been able to demonstrate its claimed high correlation to the SUS [21].

Post-test questionnaires can give useful information about the overall usefulness of a product, but it is not always as helpful in pointing out where issues exist. Instead, a post-task questionnaire can be used to find issues. Many post-task questionnaires are Likert scales. They are quick to fill out, which is important if the usability test consists of many tasks. However, Likert scales are close ended, and can suffer from ceiling and floor effects [23]. Two other questionnaires that could solve this issue are the Usability Magnitude Estimation (UME) and the Subjective Mental Effort Question (SMEQ) [23]. The UME allows users to create their own scales to indicate how difficult a task was. The SMEQ has the participant draw a horizontal line across a vertical line. The further the line is from the bottom, the more difficult the task. Both of these post task questionnaires allow users to give more exact responses than a Likert scale. Both the UME and the SMEQ, as well as Likert scales like the After-Scenario Questionnaire (ASQ), are considered to be reliable. SMEQ scores are the most closely correlated with the SUS, and allows participants a large range in which to place their response. Likert scales like the ASQ follow pretty closely to the SMEQ. The UME is found to be the least reliable,

as participants are sometimes confused on exactly how the scoring works. The UME is probably not a good alternative to Likert scales, while the SMEQ could potentially give more reliable results [21]. However, post task questionnaires do not always accurately reflect a participant's actual experience. After the task is completed the participant may downplay negative events that occurred [24]. A potential solution to this is concurrent questionnaires. The participant is asked questions while they are performing the task. It has been found that participants will place more emphasis on problems they are having and give more detail on what the issues are. Also, scores for concurrent questionnaires can vary from scores for post task questionnaires [24].

Chapter 3

Implementation

3.1 Summary of Requirements

Many of the basic requirements have already been implemented in EyeCalc 1.0 and are included in VisualOCV. These features have been translated into C# with .Net Standard 2.0 and OpenCV 4.0.1. This potentially allows the back-end library to run on any device that both OpenCV and .Net Core support. This project specifically focused on Windows 10, but it has been implemented so that it is easily portable to other platforms. This project also only focused on the computer vision functionality of OpenCV and not its machine learning functions.

When VisualOCV is opened, the user is presented with a blank workspace. The user can either load an existing algorithm or use the current blank workspace. On the left side of the screen a list of selectable operators is visible. The user can also double click the workspace to open a popup window that also contains a list of operators. Each operator will represent some function or object. These can be an OpenCV function or object, a C# primitive, a C# operator, or a user-defined function or object. When added to the workspace, the operator displays its name

and all of its inputs and outputs. Each input will be labeled and have a control that the user can modify. Outputs of operators can be connected to the input of other operators, which can later be removed if the user wants. When an operator's inputs are updated it will immediately process new outputs. If the operator runs into an error, it will indicate this to the user by changing the title bar to red, and allow the user to see an error message. While in an error state the operator is disabled, and no data will flow through it until the user resolves the issue. The user is also able to add operators to VisualOCV without needing access to VisualOCV's source code. The user has basic editing capabilities such as copying, cutting, pasting, deleting, and moving operators. The user can also group a set of operators into a single custom operator called a macro, which can be used in other algorithms or shared with other users.

More detailed requirements can be found in [Appendix A](#).

3.2 Development Approach

VisualOCV is divided into two major sub-projects, the back-end library and the GUI. The back-end library contains most of the logic of the application, such as:

- processing operator inputs
- passing data between operators
- loading operators from XML files
- saving and loading algorithms
- saving and loading macros
- copying, cutting and pasting operators

Since the back-end library has been developed with .Net Standard 2.0 it can potentially run on multiple OSs; however, only Windows 10 was specifically used during this project. Using a cross-platform framework for the GUI is preferable, but a careful examination of various available frameworks reveal that they are not yet mature, missing functionality, or do not provide a native look and feel. Instead, WPF was used to develop the GUI, for the following reasons:

- WPF easily provides a native look and feel for Windows applications.
- WPF provides more control over the GUI.
- WPF is officially supported and extensively documented by Microsoft.
- WPF has a significant amount of community adoption.
- WPF provides better performance.

3.3 Task Delineation

This project was divided into 20 two week sprints, each representing about 20 work hours. Table 3.1 gives a description of each sprint, how long it was estimated to take, and how long it actually took.

3.3.1 Back-end

The back-end library was the first part of the project to be developed. Much of the basic functionality of VisualOCV can already be found in EyeCalc 1.0 and has been translated into C#. It was created as a Class Library with .Net Standard 2.0 and utilizes Emgu CV-4.1.0, a cross platform .Net wrapper for OpenCV-4.1.0 that runs on Windows, Linux, Mac OS X, iOS, Android, and Windows Phones. Emgu CV

Table 3.1: Sprints

Sprint	Description	Estimated Hours	Actual Hours
1	Load operators from and operator communication	22	42.6
2	Cycle prevention, copy, cut, paste, and module loading	21	23.9
3	GUI communication, manage algorithm	22	30.4
4	Macros	22	27.4
5	File menu, toolbar, and operator list	24	29.6
6	Draw operator	22	16
7	Draw operator and image popup window	20	23.7
8	Dark theme and draw operator connections	15	13.6
9	Operators for C# operators and primitives	22	3.7
10	Image, video, and webcam operators	15	17.6
11	Documentation	20	33
12-15	Operators	75	66.3
16-17	Testing and debugging	40	41
18	Usability Testing	20	14.6
19	Testing and debugging	20	14.19
20	Acceptance Testing	20	3.1

is included with the application so users will not need to download it separately. Since the back-end library was developed with .Net Standard it is possible to make it cross platform, but only Windows 10 has been used in this project.

3.3.1.1 Creating New Operators

The principle component of the back-end library is the logic behind the operators. Operators essentially represent some function, which the operator directly calls. EyeCalc 1.0 provides only 83 operators and more can only be added by modifying the source code. This makes it difficult to add and distribute new operators for EyeCalc 1.0. Using reflection gives the parameters and the return type of a function, and can dynamically invoke that function. Thus, we use reflection to automatically generate operators. Users are able to create XML files which identify the operator's corresponding function, the name of the operator, and what controls should be used for each input and output. These XML files are stored in a folder in the same directory as the VisualOCV executable. The operators can be organized into sub-folders in this folder, which is how they are organized in the operators list. For the operators to access the code to invoke the function, the libraries containing the code must be accessible by VisualOCV. There is another folder in the same directory as the VisualOCV executable where users can place these libraries. This will allow for quick and easy implementation of new operators without needing to modify VisualOCV's source code. Figure 3.1 shows what these XML files look like.

It is not ideal for users creating operators to have to use the full assembly name of types whenever they need to reference it. This is because it can cause VisualOCV to fail to find the type depending on what the .Net target is or what version of a DLL is being referenced. For example, VisualOCV's back-end library is targeting .Net Standard 2.0 which allows it to run with any .Net Framework or .Net Core application that can use .Net Standard 2.0. However, .Net Framework uses different assemblies than .Net Core. The above XML examples load types from the .Net Core assemblies and thus won't work if a .Net Framework application (such

```

1 <File>
2   <Version version = "1" />
3   <Dlls></Dlls>
4   <Operator>
5     <Type AssemblyQualifiedName = "System.String, System.Private.CoreLib, Version=4.0.0.0, Culture=
6       neutral, PublicKeyToken=7cec85d7bea7798e" />
7     <Method Name = "StartsWith" param1Type = "System.String, System.Private.CoreLib, Version=4.0.0.0,
8       Culture=neutral, PublicKeyToken=7cec85d7bea7798e" />
9     <ID ID="StringStartsWith" />
10    <OperatorName Name = "String Starts With" />
11    <OperatorInputs>
12      <Input>
13        <Name Name = "String" />
14        <Type AssemblyQualifiedName = "System.String, System.Private.CoreLib, Version
15          =4.0.0.0, Culture=neutral, PublicKeyToken=7cec85d7bea7798e" />
16        <Control Control = "Textbox" />
17      </Input>
18      <Input>
19        <Name Name = "String To Check For" />
20        <MethodParamIndex Index = "0" />
21        <Control Control = "Textbox" />
22      </Input>
23    </OperatorInputs>
24    <OperatorOutputs>
25      <Output />
26      <Name Name = "Starts With?" />
27      <Control Control = "Checkbox" />
28    </OperatorOutputs>
29  </Operator>
30 </File>

```

Figure 3.1: This shows what the operator XML files look like. The <Dlls> tag informs VisualOCV what DLLs contain the methods the operators represent. After this tag each operator is defined by identifying the type containing the method and the method the operator represents. The operator's name, inputs, and outputs follow.

as a Window's GUI) tries to use it. Because of this, VisualOCV provides another XML configuration file for managing types

3.3.1.2 Managing Types

For VisualOCV to effectively use a type, it will need to know various pieces of information about it. One piece of information is the possible assemblies the type can be loaded from. This allows VisualOCV to resolve the correct assembly when the application starts. The user can define an id for the type which can then be used in the operator and validator configuration files instead of an assembly-qualified name to address the issue mentioned in the previous section. The file

```

1      <File>
2          <Version version = "1" />
3          <Dlls></Dlls>
4          <Type>
5              <ID id1 = "C#Int" id2 = "CSharpInt" />
6              <AssemblyQualifiedName
7                  AssemblyQualifiedName1 = "System.Int32, System.Private.CoreLib, Version=4.0.0.0,
8                      Culture=neutral, PublicKeyToken=7cec85d7bea7798e"
9                  AssemblyQualifiedName2 = "System.Int32, mscorlib, Version=4.0.0.0, Culture=neutral,
10                      PublicKeyToken=b77a5c561934e089" />
11              <SaveMethod ContainingType = "EyeCalcBackend.Types.TypeLoadSaveMethods,
12                  EyeCalcBackend, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" Name = "SaveInt" />
13              <LoadMethod ContainingType = "EyeCalcBackend.Types.TypeLoadSaveMethods,
14                  EyeCalcBackend, Version=1.0.0.0, Culture=neutral, PublicKeyToken=null" Name = "LoadInt" />
15          </Type>
16      </File>

```

Figure 3.2: This shows how to define multiple possible DLLs to reference an integer from, and save and load methods for an integer. Note that it has two attributes for `AssemblyQualifiedName`, one references `System.Private.CoreLib` for .Net Core and the other references `mscorlib` for .Net Framework.

will also allow the user to specify save, load and clone methods for the type so that VisualOCV knows how to save and load the type when saving and loading an algorithm and how to clone data when it is passed between operators. Instead of specifying each method individually, it is possible for each method to be placed inside a single helper class and the XML file only needs to reference the class. If the method signatures are correct, VisualOCV will be able to automatically find the methods. In some cases, an object will need to cast or convert to another type of object. VisualOCV will try to do this automatically; however, it cannot know how to do every type conversion. If a helper class is used, then convert methods can also be used to tell VisualOCV how to convert the object into another type of object. VisualOCV will provide these files for C# primitives and some OpenCV types. Figure 3.2 gives an example of what one of the files looks like.


```

1 <Validators>
2   <Version version = "1" />
3   <Dlls>
4     <Dll path = "Validators.dll" />
5   </Dlls>
6   <Validator>
7     <Name Name = "IntLessThan" />
8     <Type AssemblyQualifiedName = "Validator.Validators, Validator, Version=1.0.0.0, Culture=neutral,
9       PublicKeyToken=null" />
10    <Method Name = "IntLessThan" param1Type = "System.Int32, System.Private.CoreLib, Version=4.0.0.0,
11      Culture=neutral, PublicKeyToken=7cec85d7bea7798e"
12    param2Type = "System.Int32, System.Private.CoreLib, Version=4.0.0.0, Culture=neutral, PublicKeyToken
      =7cec85d7bea7798e" />
11   </Validator>
12 </Validators>

```

Figure 3.3: An example of what defining validators looks like. The file is very similar to an operator XML file. The type containing the method must be specified and the DLL containing the type must also be specified.

```

1 <Input>
2   <Name Name = "input" />
3   <MethodParamIndex Index = "0" />
4   <Control Control = "Slider" />
5   <Validator Name = "IntLessThan" max = "100" />
6   <Validator Name = "IntGreaterThan" min = "0" />
7 </Input>

```

Figure 3.4: This shows how a validator can be assigned to an operator's input in the operator XML file.

3.3.1.3 Operator Validators

Users may want to specify what is considered a valid input to the operators they are creating. Another set of XML files can be used to specify methods that can be used as validators. Similar to operators, the file must state the DLLs that contain the code for the validators and each validator needs Type and Method tags to identify the method. This allows users to create their own validators if an appropriate one does not already exist. While the method that an operator represents can have any number and type of parameters and any kind of return type, validators cannot. A method to be used as a validator must return a tuple containing a bool indicating if the validator failed or not and a string containing an error message. The first parameter must also be the object being checked. Figures 3.3 and 3.4 give an example of how validators are defined and used.

3.3.1.4 Operator Communication

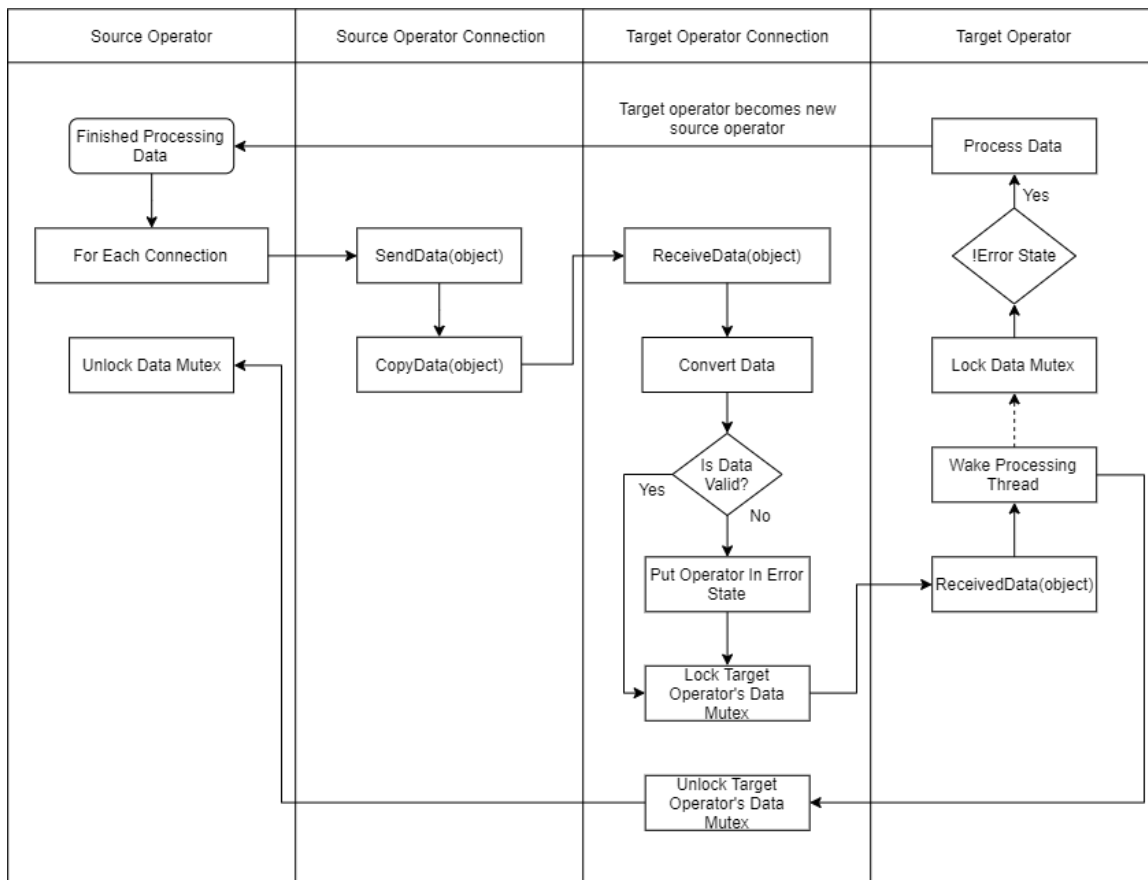


Figure 3.5: This diagram indicates how each operator communicates with each other. When an operator is done processing its data, it will send its output to each operator that output is connected to. The receiving operator will store that data, then wake up its processing thread to process the data. After the data has been processed, the operator will send its outputs to its connected operators.

The back-end library implements the communication between each operator. Figure 3.5 shows how operators communicate with each other. Each operator has a thread that will process its data. If an operator's input is changed, the operator wakes the processing thread, validates the data, and processes the new data if it is valid. After an operator finishes processing its inputs, it passes a copy of its output to each operator to which it is connected. Each operator will have a mutex

to protect inputs and outputs from race conditions.

3.3.1.5 Marcos

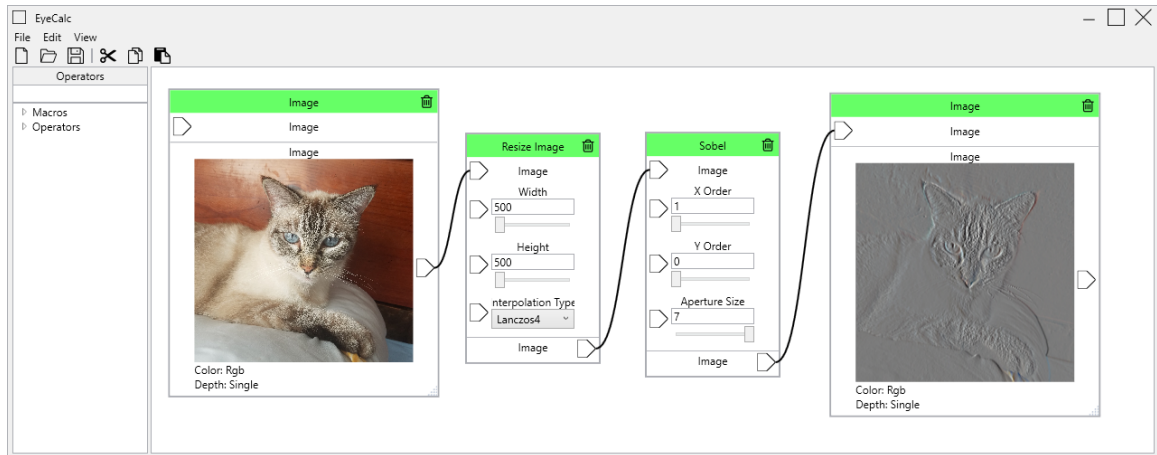
Once a user creates an algorithm, they may want to use it in another algorithm. Users can put an entire algorithm into a single operator called a macro. Developing a macro will be the same as making an algorithm. Users have input and output operators that are used to identify what the inputs and outputs of the macros are. Macros can be saved in a folder located in the same directory as the executable. The macros are displayed and organized in the operator list in the same way they are stored in the folder.

3.3.1.6 Synchronization

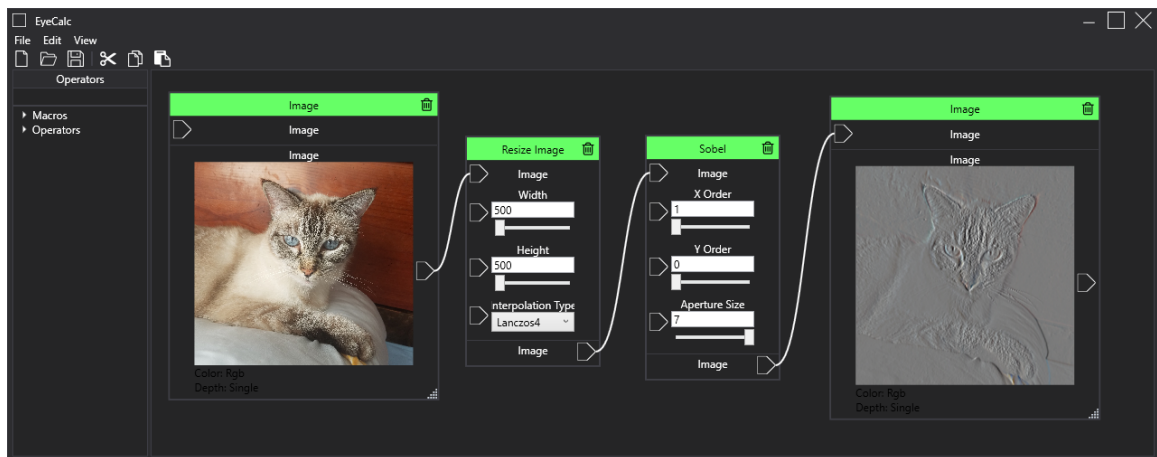
EyeCalc 1.0 is unable to guarantee synchronized data along parallel paths. This has been resolved by tracking how many operators are running, and preventing the root operators from sending data until all operators have completed processing. An operator is considered running when it receives new data and is no longer running after it sends a copy of the output to each of the operators to which it is connected.

3.3.2 GUI

After the back-end library was completed work began on the GUI. Windows Presentation Foundation (WPF), a native GUI framework for Windows, was used to create the GUI. As shown in Figure 3.6 the application has a menu and tool bar containing features such as saving and loading an algorithm and copying, pasting, and cutting operators. A list of operators is provided on the left side of



(a)



(b)

Figure 3.6: This shows what the application currently looks like in its light theme (a) and dark theme (b). The top contains the menu and toolbars. The left side of the screen holds the list of operators. In the middle of the screen is the main workspace. Inside the main workspace is the operators.

the screen. The list is a tree-view with the operators grouped into various modules. A search bar is also provided above the list. To add an operator to the workspace, the user can double click, or click and drag an item from the list. The workspace takes up the majority of the screen, and is where the users create their algorithms. The operators are drawn here, including their inputs and outputs, along with the input's and output's names and controls. The title bar of the operator displays the

name of the operator and will turn red if the operator runs into an issue when attempting to process its inputs, such as receiving an invalid input. VisualOCV also provides a popup window to better view images. By double clicking on an image operator that is displaying an image, a new window will popup containing a larger version of the image. From here, the user is able to pan around and zoom in and out, allowing the user to get a better look at the image. This window also provides a save option to allow the user to save the image. The image in this window updates when the image operator receives a new image to display.

3.3.3 Adding Operators

The remaining time was used to add more operators, for a total of 141 operators. The more operators available to the user, the more useful EyeCalc will be. Most operators will represent an OpenCV function or object. However, there are operators for all built-in C# primitives except the object type, and for some C# operators. Table 3.2 shows what C# operators will be available.

Table 3.2: Supported C# Operators

Operator	Description
$x[]$	Index the operand
$!x$	Logical negation
$(T)x$	Type casting
$x*y$	Multiply x and y
x/y	Divide x by y
$x\%y$	x mod y
$x + y$	Add x and y
$x - y$	Subtract y from x
$x < y$	Less than
$x > y$	Greater than
$x \leq y$	Less than or equal
$x \geq y$	Greater than or equal
$x == y$	Equality
$x != y$	Not equal
$x \& y$	Logical AND
$x \wedge y$	Logical XOR
$x y$	Logical OR

3.3.4 Operator Constructor

While adding operators to VisualOCV an operator constructor tool was created to generate the basic form of the XML needed for that operator. This tool allows a user to load an assembly, select a class, and then select one of the class's methods. Once a method is selected, the right side of the window will be populated with information about the method, as well as the XML needed to create the operator. This XML is only enough to load the operator into VisualOCV; some parts of it will still need to be edited by the user, such as the operator name, the input and output names, and controls.

3.4 Usability Survey

Near the completion of the project a usability survey was given to determine the usability of the application. Ten participants were asked to perform a set of tasks, and after each task, they answered a seven-point Likert-scale question about the difficulty of the task. A Likert scale was used instead of an Usability Magnitude Estimation (UME) and Subjective Mental Effort Question (SMEQ) due to its reliability and simplicity. Each question also has a place where participants can put comments. After the tasks are completed each participant filled out a System Usability Scale (SUS) to help determine the overall usability of the application. The usability survey can be found in Appendix B

3.5 Final Deliverables

An application installer and VisualOCV's source code are available on the project's [GitLab page](https://gitlab.com/johnboggess/VisualOCV) (<https://gitlab.com/johnboggess/VisualOCV>). The project will also be documented for anyone who wants to use the source code. The documentation will be put on the project's GitLab wiki.

Chapter 4

Testing & Evaluation

4.1 Acceptance

It is important that all the requirements are met once the project is finished. To ensure this, an Acceptance test was given and can be seen in Appendix C. The test was given to one faculty member and one graduate student. The test gives a list of different features and requirements and asks if participants are able to perform or confirm that the requirements met in the final project. One question was answered no, leaving approximate 99% of the questions answered yes, which exceeds the desired 80%.

4.2 System

The overall functionality of VisualOCV was tested to make sure the whole system functions properly. Table 4.1 gives an overview of the usability and performance tests done.

Table 4.1: System Tests

Tests	Goal	Result
System Usability Scale	Average score of at least 70	85
Time to Load Operators	Less then 1 seconds for 50 operators	61 ms
VisualOCV vs C# Performance	VisualOCV should not take 50% longer than C#	10%

4.2.1 Usability

Ten individuals (potential users, graduate students and upperclassmen) completed the usability test found in Appendix B. The testing was performed over multiple sessions with as many participants as could be gathered at once. The test had the participants perform various tasks that covered most of the features available

Table 4.2: Usability Test Results

Task	Average Score
1	6.7
2	6.3
3	6.4
4	6.5
5	6.5
6	6.5
7	6.5
8	6.4
9	6.5
10	6.9
11	6.0
12	6.7
13	6.6

to them and most of the system's requirements. Each task has a seven point Likert-scale indicating if the task can be performed and how easily the participants were able to complete it. The usability test also utilized a System Usability Scale to determine overall usability of the application. The test is considered successful if it meets the following criteria:

- The participants are able to finish each task without running into any serious issues.
- The average score for each task is at least a five.
- The System Usability Score is at least seventy or higher.

All 10 participants were able to complete the test, with an minimum average task score of 6 (with a standard deviation of 1.3), and an average System Usability Score of 84.3 (with a standard deviation of 12.4). The results for each task can be seen in Table 4.2.

4.2.2 Performance

Performance is critical to various parts of VisualOCV, and thus performance testing was done to ensure the application runs appropriately. Performance tests were done on a Windows 10 machine with an Intel Core i7-4810MQ processor, NVIDIA Quadro K2200M graphics card, HGST HTS7210 7200 RPM hard drive , and 24 GB of RAM.

One test was to check how quickly VisualOCV is able to load modules as there may be many modules providing a large number of operators to VisualOCV. A dummy module with 50 dummy operators was loaded and the time to load them was recorded. After running this test 15 times it was found that the 50 operators

were loaded within 61 ms on average (with a standard deviation of 4.1 ms), well within the one second goal.

It is also critical that the overhead of operators passing and validating data does not take a significant amount of time. A simple green screen algorithm was created in both C# and in VisualOCV and the amount of time it took each to complete a single frame was recorded. This time was used to determine how much VisualOCV's overhead affects performance. The original goal for VisualOCV to take no more than 50% longer than the C# algorithm to complete. This test was run 15 times and it was found that VisualOCV took on average 66 ms (with a standard deviation of 6.8 ms) to complete the algorithm and C# took an average of 60. ms (with a standard deviation of 0.8 ms) to complete. The average ratio between the VisualOCV's time and the C#'s time was 1.1 (with a standard deviation of 0.11), meaning on average VisualOCV took approximately 10% longer.

4.3 Integration

Various integration tests have been performed to make sure VisualOCV functions correctly. Table 4.3 gives an overview of the tests performed

VisualOCV must correctly import modules, or there will be no operators

Table 4.3: Integration Tests

Test	Successful
Correctly imports modules	✓
Save and load algorithm	✓
Correctly process and pass data	✓
Macros function correctly	✓

available to the user. An automated test was created that imports a module with a known set of operators. The module `Operators.Constructors`, which contains operators for creating C# primitives, was loaded and found to correctly load all its operators.

VisualOCV must be able to save and load algorithms correctly. This is tested with an automated test that saves a known algorithm and then loads it. Saving a simple image processing algorithm and loading it again results in the same algorithm.

The operators need to be able to correctly process data and pass the results to any connected operators. An automated test compares the result of an algorithm created with operators and the result of the same algorithm written in C# to make sure they are the same. A simple image processing algorithm was created with both VisualOCV and C#, and given the same inputs. It was found that the results were the same.

A test has been performed to to make sure that macros function correctly. A macro was created and then loaded into another algorithm. The macro result is then compared to the result of the same algorithm made of operators, which should be the same. It was found that the operators and the macro produced the same results.

4.4 Module

There are various modules to test to ensure that VisualOCV functions correctly. Table 4.4 gives a overview of the module tests performed.

Copying, cutting, and pasting operators were tested using an automated test that was created by copying and pasting a set of known operators and checking

Table 4.4: Module Tests

Test	Successful
Copy, cut and paste	✓
Searching the operator list	✓
Cycle prevention	✓
Passing data between operators	✓

that the new operators match the copied operators. It was found that the pasted operators matched the operators from which they were copied.

The search bar on the operator list was tested to ensure it functions properly. This was tested by giving various search terms and checking the results. If the results of the search match the search term then this test was considered successful. The test takes a random sub string of a random operator's name and uses that as the search term. The search results are checked to make sure it only contains operators whose name contains the search term and checks all the existing operators to make sure all operators whose name contains that search term appear in the search results. This test was repeated twenty times and it was found that the search results contained only operators whose name contains the search term, and that the results contain all operators whose name contains the search term.

Since it is also important that loops can not be created, this was tested. In an automated test, several operators were loaded and attempted to be connected in various ways that would cause a loop; no loops were created.

Operators must also be able to pass data between one another. A test for this was done automatically by sending known data from one operator to another. The operators that were selected for this test were selected due to the different ways that their data needed to be handled when passed between each other. It

was found that the receiving operator received the correct data from the sending operator.

4.5 Unit

The testing framework used for unit testing was NUnit [25]. Unfortunately NUnit does not provide code coverage reports, so Coverlet was used to for this. Currently 82% of the code base has been covered by testing, which exceeds the 80% code coverage goal.

Chapter 5

Conclusion

VisualOCV is a tool to help users easily understand and prototype computer vision algorithms. This project translated EyeCalc 1.0 from C++ to C# and used more modern technologies. VisualOCV contains most of the same features as EyeCalc 1.0 and adds some new features. It is possible for users to create and distribute new operators without needing to modify the application's source code. Users are also be able to group an algorithm into a macro, which act as an operator that can be used in other algorithms. Development of the application focused on Windows 10, however the back-end library was created with .Net Standard, which should make it easy to port other operating systems.

5.0.1 Future Work

There is still work that can be done to improve VisualOCV. Providing GUIs for other platforms would allow VisualOCV to reach a larger audience. A preferences menu could be created that allows users to set where files such as operators, macros, and crash logs are located. The view menu could also have some zoom options, such as zoom to fit. Once an algorithm has been created a user may want

to write it in another language. It could be useful to have VisualOCV generate pseudo code that would make it easier to translate. More operators can always be added to VisualOCV, which will make it more useful.

Appendix A

Requirements Specification

A.1 Application Requirements

A.1.1 Operator Requirements

- Operators shall display their name.
- Operator outputs shall be able to be connected to the inputs of other operators.
- All operators shall display their outputs.
- All operator inputs and outputs shall be labeled.
- An operator's input's controls shall be disabled if an another operator's output is connected to that input.
- The user shall be able to right click an operator to open a context menu.
- The user shall be able to left or right click an operator to select it.
- The user shall be able to shift+click or ctrl+click to select multiple operators.

- The user shall be able to click and drag while holding shift to select operators.
- The user shall be able to deselect operators by clicking somewhere on the workspace.
- The user shall be able to deselect operators by selecting a new operator.
- The user shall be able to shift+click or ctrl+click a selected operator to deselect it.
- The user shall be able to delete selected operators by pressing the delete key.
- Operators shall provide a delete context menu option for deleting operators.
- Operators shall have a delete button on the operator's title bar.
- The user shall be able to save the image displayed by an image operator.
- The user shall be able to double click the image on an image operator to see an enlarged version of the image.
- The user shall be able to zoom into the enlarged image.
- The user shall be able to pan around the enlarged image.
- The enlarged image should update if the operator's image changes
- The user shall be able to move selected operators by clicking and dragging one of the selected operators.
- The user shall be able to copy, cut, and paste operators.
- Operators shall provide copy, cut, and paste context menu options.
- The main window's toolbar shall provide copy, cut, and paste options.

- The user shall be able to use ctrl+c, ctrl+x and ctrl+v hotkeys to copy, cut, and paste.
- The application shall provide image, video file, and webcam operators.
- Operators shall be able to catch most errors that occur while processing their data.
- An Operator's title bar shall turn red indicate to the user it is in an error state.
- The user shall be able to see what the error is that an operator has encountered.
- The user shall be able to double click the title bar of an operator in an error state to see what the error is.
- Operators shall provide a Show Error context menu option to see what error the operator encountered.
- Operators shall validate their inputs and throw an appropriate error if the inputs are not valid.
- Operators shall display a generic error message if an unknown error has occurred.
- An operator will throw a type mismatch error if another operator gives it a type different than what is expected.
- Operators in an error state shall be disabled.
- The application shall have operators for most C# primitives.

- The application shall provide operators for most C# operators.
- Users should be able to create operators from existing code.
- Operators shall be organized into modules and sub-modules.

A.1.2 Connections

- Connections between operators shall be displayed as a Bézier curve.
- The user shall be able to left click a connections to select it.
- The user shall be able to shift+click or ctrl+click to select multiple connections.
- The user shall be able to ctrl+click a selected connection to deselect it.
- The user shall be able to deselect connections by clicking somewhere on the workspace.
- The user shall be able to deselect connections by selecting a new connection.
- The user shall be able to delete selected connections by pressing the delete key.
- The user shall be able to disconnect two operators by double clicking the pins they are connected to.

A.1.3 Macros

- The user should be able to create macros.
- The user should be able to edit existing macros
- The applications file menu should provide a "Save As Macro" option

- The user should be able to group macros into modules.
- The application should provide input and output operators that represent the macro's inputs and outputs.
- The user should be able to specify the name of inputs and outputs of the macro.
- The application shall provide an exception operator to allow error detection in macros.

A.1.4 Workspace

- The user shall be able to save the current workspace.
- The user shall be able to open a workspace.
- The user shall be able to create a new workspace.
- The user shall be able to move the workspace by clicking and dragging.
- The application shall provide a list of operators on the left side of the workspace.
- The application shall provide a pop up list of operators by double clicking the workspace.
- The operators in the operator list shall be grouped by modules.
- The application shall allow the user to search the list of operators.
- The list of operators shall be alphabetical.

- The user shall be able to double click an item in operator list to add to workspace.
- The user shall be able to click and drag from list on side of screen to add operator.

A.1.5 General

- The application shall have light and dark themes.
- The application shall utilize OpenCV 4.0.1.
- .Net Standard 2.0 shall be used for the back-end library.
- The back-end project should hold all logic.
- The application shall support most of the operators the original EyeCalc supports.
- The main window's toolbar shall provide options to create a new workspace and save or open a workspace.
- The main window's file menu shall provide New, Open, Save, and Save As options.
- The main window's edit menu shall provide Copy, Cut, Paste options.
- An error shall be displayed to the user if a module fails to load. The application shall continue running without loading that module.

Appendix B

Usability Survey

EyeCalc is an application that provides a dataflow programming environment for learning about and prototyping image processing algorithms. EyeCalc provides various operators, each of which represent a function, that can be combined in various ways to produce image process algorithms. This survey steps you through the process of creating a simple green screen algorithm and is intended to measure the usability of the application. After performing each task below use the Likert scale to indicate how difficult (lower) or easy (higher) the task was. If you have any issues or suggestions, please place them in the comments section after each task. After the tasks are completed there is another short survey on the last page about your overall experience with EyeCalc.

1. Add two “Image” operators, two “Color” operators, and an “In Range (Colors)” operator to the workspace.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

2. Load Foreground.jpg (in the folder with executable) into an “Image” operator and set all the color values of one of the “Color” operators to 255.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

3. Connect the “Image” operator to the “In Range (Colors)” operator, connect the two “Color” operators to the “In Range” operator (connect the one with the 255 color values to the input labeled “Higher”), and connect the output of the “In Range (Colors)” operator to the other “Image” operator.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

4. Adjust the color values of the “Color” operator connected to “Lower” so that the person is white and the background is black.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

5. Add the “Multiply” operator from the OpenCV module and input the original image and the black and white image. The operator should be in an error state, check the error message.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

6. Disconnect both images from the “Multiply” operator and use “Convert” operators to convert both images to RGB with a depth of Float. Connect the converted images to the “Multiply” operator.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

7. Copy and paste the two “Convert” and “Multiply” operators below the existing operators.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

8. Use a “Not” operator to invert the output of the “In Range (Colors)” operator and load in Background.jpg (in the folder with executable) into a new “Image” operator. Connect both to the pasted “Convert” operators.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

9. Use “Add Images” to add the results of the two “Multiply” operators and connect the output to a new “Image” operator.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

10. Save the resulting image.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

11. Insert “Macro Input” operators between each of the source images and assign them reasonable names. Replace the “Image” operator showing the result of the algorithm with a “Macro Output” operator.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

12. Save the workspace as a macro (save it in the default location given) and create a new workspace.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

13. Add the new macro and the two source images to the workspace. Connect the images to the macro and connect the output to another image operator.

Overall, how difficult was this task?

Very Difficult						Very Easy
1	2	3	4	5	6	7

Comments:

Based on your experience with EyeCalc, rate the following statements. If you are not sure how to respond, mark "3".

Strongly
Disagree

Strongly
Agree

1. I think that I would like to use this system frequently.

1	2	3	4	5
---	---	---	---	---

2. I found the system unnecessarily complex.

1	2	3	4	5
---	---	---	---	---

3. I thought the system was easy to use.

1	2	3	4	5
---	---	---	---	---

4. I think that I would need the support of a technical person to be able to use this system.

1	2	3	4	5
---	---	---	---	---

5. I found the various functions in this system were well integrated.

1	2	3	4	5
---	---	---	---	---

6. I thought there was too much inconsistency in this system.

1	2	3	4	5
---	---	---	---	---

7. I would imagine that most people would learn to use this system very quickly.

1	2	3	4	5
---	---	---	---	---

8. I found the system very cumbersome to use.

1	2	3	4	5
---	---	---	---	---

9. I felt very confident using the system.

1	2	3	4	5
---	---	---	---	---

10. I needed to learn a lot of things before I could get going with this system.

1	2	3	4	5
---	---	---	---	---

11. Overall, I would rate the user-friendliness of this product as:

Worst Imaginable	Awful	Poor	OK	Good	Excellent	Best Imaginable
------------------	-------	------	----	------	-----------	-----------------

Appendix C

Acceptance Survey

Test	Yes	No	Comments
Are operators in the operator list organized into modules?			
Can operators be added by clicking and dragging from the operator list?			
Can operators be added by double clicking an item in the operator list?			
Can operators be added by pressing enter when an operator is selected in the operator list?			
Does the operator list have a search bar?			
Does the search find an operator that matches the search term?			
Does the search results update as the user types?			
Can operators be connected?			
Can operators be disconnected by double clicking the connection pin?			
Can an operator be moved by click and dragging the operator's title bar?			
Is an operator selected when clicked on and other selected operators unselected?			
Can multiple operators be selected by clicking and dragging?			
Does clicking on an operator while holding ctrl or shift allow multiple operators to be selected?			
Does clicking anywhere in the workspace cause selected operator to become unselected?			
Does clicking on a selected operator while holding ctrl or shift unselect just that operator?			
Does pressing the delete key delete selected operators			
Does the operator's context menu provide a delete option?			
Does clicking the delete button on the operator's title bar delete the operator?			
Do operator parameters have controls that can be modified?			
Are the values of an operator's parameters displayed?			
Does an operator's output change as its inputs change?			

Can images be loaded into the image operator?			
Can an image from the image operator be saved?			
Does double clicking the image on the image operator open a popup window with the image in it?			
Can the image in the popup window be zoomed in on?			
Can the image in the popup window be panned			
Does the image in the popup window change as the image on the operator changes?			
Does the file menu provide copy, cut, and paste options?			
Does the toolbar provide copy, cut and paste options?			
Does an operator's context menu provide copy, cut and paste options?			
Does the normal copy, cut, and paste hotkeys work?			
Can the workspace be moved (panned)?			
Can the workspace be saved?			
Can a workspace be loaded correctly?			
Can the new option be used to create a new workspace?			
Can an algorithm be saved as a macro?			
Can a macro be used in another algorithm?			

Bibliography

- [1] OpenCV. (2018, dec) Introduction. [Online]. Available: <https://docs.opencv.org/4.0.1/> (document), 2.1, 2.1
- [2] Opencv. [Online]. Available: <https://opencv.org/> 2.1
- [3] J. Boggess. (2019, may) Eyecalcold. [Online]. Available: <https://gitlab.com/johnboggess/eyecalcold> 1
- [4] M. C. V. S. T. Team. (2019, mar) Computer vision toolbox opencv interface. [Online]. Available: <https://www.mathworks.com/matlabcentral/fileexchange/47953-computer-vision-toolbox-opencv-interface> 2.2
- [5] P. Corke, *Robotics, Vision & Control*. Springer, 2017. 2.2
- [6] LabView. (2019) Ni vision opencv utilities. [Online]. Available: <http://sine.ni.com/nips/cds/view/p/lang/en/nid/213723> 2.2
- [7] OpenCV. (2015, dec) Opencv demonstrator (gui). [Online]. Available: <https://opencv.org/opencv-demonstrator-gui/> 2.2
- [8] ——. (2019, jun) Interactive visual debugging of computer vision applications. [Online]. Available: https://docs.opencv.org/3.4/d7/DCF/tutorial_cvv_introduction.html 2.2

- [9] D. Escrivá. (2014) Blendocv. [Online]. Available: <https://github.com/damiles/blendocv> 2.2
- [10] Qmlnet. (2019, jul) Qmlnet. [Online]. Available: <https://github.com/qmlnet/qmlnet> 2.3
- [11] Mono. (2019, may) xwt. [Online]. Available: <https://github.com/mono/xwt> 2.3
- [12] D. Dobrev. (2019, jul) Qtsharp. [Online]. Available: <https://github.com/ddobrev/QtSharp> 2.3
- [13] AvaloniaUI. (2019) Avalonia. [Online]. Available: <https://github.com/AvaloniaUI/Avalonia> 2.3
- [14] Mono. (2018, may) Gtk#. [Online]. Available: <https://github.com/mono/website/blob/gh-pages/docs/gui/gtksharp/index.md> 2.3
- [15] P. S. Solutions. (2019, jul) Eto. [Online]. Available: <https://github.com/picoe/Eto> 2.3
- [16] Microsoft. (2017, mar) Windows forms. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/winforms/> 2.3
- [17] ——. (2018, jan) Introduction to wpf in visual studio. [Online]. Available: <https://docs.microsoft.com/en-us/dotnet/framework/wpf/getting-started/introduction-to-wpf-in-vs> 2.3
- [18] BBVAOPEN4U. (2016, nov) Differences between native mobile and cross-platform applications. [Online]. Available: <https://bbvaopen4u.com/en/actualidad/differences-between-native-mobile-and-cross-platform-applications> 2.3

- [19] M. Dennis. (2018, dec) Native vs. cross-platform apps – you’ll be the winner. [Online]. Available: <https://www.zeolearn.com/magazine/native-vs-cross-platform-apps-youll-be-the-winner> 2.3
- [20] M. J. Garbade. (2018, aug) Native vs. cross-platform app development: pros and cons. [Online]. Available: <https://codeburst.io/native-vs-cross-platform-app-development-pros-and-cons-49f397bb38ac> 2.3
- [21] J. R. Lewis, B. S. Utesch, and D. E. Maher, “Umux-lite: When there’s no time for the sus,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’13. New York, NY, USA: ACM, 2013, pp. 2099–2102. [Online]. Available: <http://doi.acm.org.ezproxy.southern.edu/10.1145/2470654.2481287> 2.4
- [22] A. Bangor, P. Kortum, and J. Miller, “Determining what individual sus scores mean: Adding an adjective rating scale,” *J. Usability Studies*, vol. 4, no. 3, pp. 114–123, May 2009. [Online]. Available: <http://dl.acm.org.ezproxy.southern.edu/citation.cfm?id=2835587.2835589> 2.4
- [23] J. Sauro and J. S. Dumas, “Comparison of three one-question, post-task usability questionnaires,” in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI ’09. New York, NY, USA: ACM, 2009, pp. 1599–1608. [Online]. Available: <http://doi.acm.org.ezproxy.southern.edu/10.1145/1518701.1518946> 2.4
- [24] R. Teague, K. De Jesus, and M. N. Ueno, “Concurrent vs. post-task usability test ratings,” in *CHI ’01 Extended Abstracts on Human Factors in Computing Systems*, ser. CHI EA ’01. New York, NY, USA: ACM, 2001, pp.

289–290. [Online]. Available: <http://doi.acm.org.ezproxy.southern.edu/10.1145/634067.634238> 2.4

[25] NUnit. (2017) NUnit. [Online]. Available: <https://nunit.org/> 4.5

[26] SciLab. (2019) Opencv atoms. [Online]. Available: <https://atoms.scilab.org/?order=&direction=&%20key=OpenCV&id=on&title=on&description=on&platform=&%20scilab%20supported%20version=#>