Developer’s Guide

This document will help you get up and developing for UCSD’s Global TIES Digital Vision Screening project.

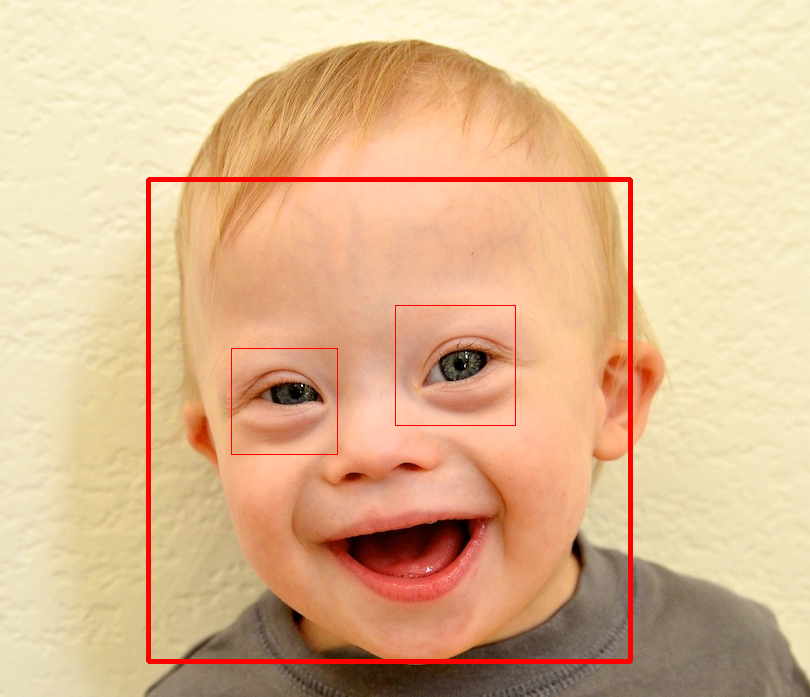


Table of Contents

[Project Overview 3](#_Toc382671169)

[Summary 3](#_Toc382671170)

[Ultimate Goal 4](#_Toc382671171)

[Subject Matter 5](#_Toc382671172)

[Get Coding! 5](#_Toc382671173)

[Technologies Overview 6](#_Toc382671174)

[Front End 6](#_Toc382671175)

[Back End 6](#_Toc382671176)

[Environment Setup 6](#_Toc382671177)

[Windows 6](#_Toc382671178)

[Linux 7](#_Toc382671179)

[Mac 7](#_Toc382671180)

[Code Structure 8](#_Toc382671181)

[Front End 8](#_Toc382671182)

[Backend 8](#_Toc382671183)

[Hardware 9](#_Toc382671184)

[Github 101 9](#_Toc382671185)

[Resources 10](#_Toc382671186)

[Documents (in the /doc folder on github) 10](#_Toc382671187)

[User Stories (not updated in Wi14) 10](#_Toc382671188)

[Requirements (not updated in Wi14) 10](#_Toc382671189)

[Class Diagrams 10](#_Toc382671190)

[Links/Quick Reference 11](#_Toc382671191)

[Technologies 11](#_Toc382671192)

[Editors 11](#_Toc382671193)

[Git 11](#_Toc382671194)

[Eye Detection 11](#_Toc382671195)

[Pupil and Iris Detection 11](#_Toc382671196)

[Sclera/Crescent Detection 11](#_Toc382671197)

[Software Concepts 11](#_Toc382671198)

[Miscellaneous 12](#_Toc382671199)

# Project Overview

## Summary

The DVS Image Analysis team works on developing algorithms that perform computational image analysis on a patient's image to determine the presence of three types of vision disorders: amblyopia, strabismus, and cataracts. Amblyopia, or more commonly referred to as "lazy eye," is the loss of or reduced vision in one eye. It is the most common cause of visual impairment in children. Strabismus, more commonly known as "crossed eyes", refers to the misalignment of both eyes. This prevents the eyes from working together, resulting in vision impairment, faulty depth perception, double vision, and, if left untreated, amblyopia. Cataracts is a clouding of the eye that can impede vision.

For more information about the three types of vision disorders we are working to identify, the following links are useful resources.

Amblyopia: <http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0002009/>

Strabismus: <http://www.ncbi.nlm.nih.gov/pubmedhealth/PMH0001999/>

Cataracts: <http://www.nei.nih.gov/health/cataract/cataract_facts.asp>

The Image Analysis(Backend) team works on one of the essential parts of the Image Analysis application; primarily the detection of the pupil in an image of the patient's eye and from there assessing the amount of the reflection in the eye caused by the [Bruckner Reflex test](http://abcd-vision.org/vision-screening/bruckner.html) (in other words, crescent detection) to ultimately determine whether the patient has an eye disease. The Bruckner Reflex test is highly dependent on capturing images with a strong red-eye effect. This does not, however, mean that the eye will look red in a “red eye” photo. For an example of a red eye photo with a significant crescent that is not fully “red” take a look at the photo of Arvind below:



*Above: Photo of example of a crescent on the bottom of a pupil with a “red eye” that is not fully red. Photo taken with the official DVS camera*

The backend code also uses the position of what we call the “white dot”, a reflection of the camera’s flash on the pupil, to determine if the patient has strabismus. In Arvind’s photo above you can see that his white dots are dead center in the middle of his pupil because he is looking directly at the camera.

Please also note that it is a secondary priority to use the information we detect about the pupil, crescent, and white dot to detect other diseases such as anisocoria, anisometropia, and amblyopia. We also hope, as a tertiary priority, to display information about the eye such as the pupil size (in mm) and papillary distance (in mm).

## Ultimate Goal

The setup we’ll be working with is a laptop connected to a camera. This camera will take two pictures, one horizontal and one vertical (this is to maximize the crescent effect of red eye). The picture need not look rotated to the side as long as the flash is rotated 90° between photos. This is so we can get a photo with a crescent on the bottom(or top) or the pupil and a photo with the crescent on the side of the pupil in order to normalize our determination of the crescent’s area, among other reasons. The program will then detect the child’s eyes, allow the user to confirm this detection, detect several other features including the pupil, crescent, white dot, and any cataracts present. These accurate detection of these features will be confirmed (or corrected) by the user. The program will then move on to analyze these eye features and notify the user of the application which, if any, eye disease the child may have, giving a “refer” or “don’t refer” result with details that may then be printed.

## Subject Matter

I’m going to be lazy with this section and refer you to the [EyeDiseaseDescriptions](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/EyeDiseaseDescriptions.docx) doc on github. It reviews the eye-related subject matter that we need to be familiar with. Refer there if you’re confused.

# Get Coding!

Python Editor (Canopy recommended)

GitHub (Build Software Better, together)

Download Library (OpenCV, PIL )

Clone DVS-Python Repository

Get Coding!

The following sections will outline the details of what you need to do to get set up but here’s the short list:

1. Get yourself added to the UCSD-TIES github organization
   1. Your TA or team lead will be able to do this. Ask them.
2. Clone the DVS-Python repository using git
   1. You may want to download the Github for Windows or Github for Mac GUI
3. Set up your environment (openCV, Canopy, github on labtop, etc.)
4. Ensure that the code you cloned works as intended on your local machine
5. Make a change to a file (add a comment or something)
6. Save the change, commit it, and push to the github repository (to make sure you can)
7. Take a look at the repository’s /doc folder, read documentations for the tech we’re using and *talk to your teammates* to get yourself oriented with the existing project and what you should be doing.
8. Get coding!

## Technologies Overview

We use [Python](http://docs.python.org/release/2.7.4/tutorial/index.html) (use Python 2, not Python 3 for compatibility with openCV. The most current version of Python 2 is fine, even if it’s not what’s listed below) as our coding language. The backend uses [PIL](http://www.pythonware.com/library/pil/handbook/index.htm) and [OpenCV](http://opencv.willowgarage.com/documentation/python/) libraries for image analysis. The Front End uses [wxPython](http://wxpython.org/onlinedocs.php) to develop a local GUI (this is because the client will not have internet access when they use the application). We use [github](http://github.com/UCSD-TIES/DVS-Python) for version control. We don’t have a particular editor or [IDE](https://en.wikipedia.org/wiki/Integrated_development_environment) we work in. Some use [IDLE](http://www.ai.uga.edu/mc/idle/), some [Pydev](http://pydev.org/), or whatever works and you can get to be set up with OpenCV. Take a look at [sublime](http://www.sublimetext.com/2) if you’re looking for a solid and simple editor (you can even edit multiple lines at the same time, it’s pretty neat). The Canopy IDE is highly recommended for ease of installation.

### Front End

* [wxPython](http://wxpython.org/download.php)
  + [Python 2.7.4](http://www.python.org/getit/)

### Back End

*Dependent technologies are indented*

* [OpenCV 2.4.5](http://opencv.org/downloads.html) (library)
* [NumPy](http://docs.scipy.org/doc/numpy/user/install.html) (latest revision as of 05/01/13)
  + Python 2.7.4
* [PIL 1.1.7](http://www.pythonware.com/products/pil/) (library)
  + Python 2.7.4
* Python editor (Canopy, sublime)
  + Python 2.7.4

## Environment Setup

See the [github wiki](https://github.com/UCSD-TIES/DVS-Python/wiki) for this. Expect a fair amount of craziness trying to get your setup working right. And if you get stuck or kinda stuck ask a teammate! Someone’s probably seen that error before.

### Windows

Before you try installing things the necessary libraries separately *try Canopy*. It has cut most installations from 4+ hours of headaches and error messages down to maybe one hour or so of clicking buttons. You must request an academic license [here](https://www.enthought.com/products/canopy/academic/) to get the full version of Canopy. Use your UCSD email address. Do not download Canopy Express, it will not let you install all of the packages that are required. After you have your canopy academic license sign into your account, download the full version of Canopy, open Canopy and install the necessary packages through canopy. You will then need to manually edit two lines in Pupil.py for the code to work. They are the lines that read PUPILPHOTO.jpg from disk. You will need to change the path to the relevant path for your computer. Yes, I know it’s janky. We weren’t sure how to implement relative paths for that file and if we don’t write the photo to disk we end up with errors we can’t figure out how to make go away. Having absolute paths specified is spaghetti code that will need to be fixed. After that you should be good to run the program. If you would like to run testing code for the backend set the TEST flag at the top of Controller.py to true and run Controller.py. If you would like to run the front end code set the TEST flag at the top of Controller.py to false and run main.py.

See the github wiki.

1. Install Python Editor
2. Instal NumPy
3. Install OpenCV
4. Copy openCV into the appropriate python path directory.
5. Install the Python Imaging Library
6. Set Path

### Linux

See the github wiki. And good luck.

### Mac

Before you try installing things the necessary libraries separately *try Canopy*. It has cut most installations from 4+ hours of headaches and error messages down to maybe one hour or so of clicking buttons. You must request an academic license [here](https://www.enthought.com/products/canopy/academic/) to get the full version of Canopy. Use your UCSD email address. Do not download Canopy Express, it will not let you install all of the packages that are required. After you have your canopy academic license sign into your account, download the full version of Canopy, open Canopy and install the necessary packages through canopy. You will then need to manually edit two lines in Pupil.py for the code to work. They are the lines that read PUPILPHOTO.jpg from disk. You will need to change the path to the relevant path for your computer. Yes, I know it’s janky. We weren’t sure how to implement relative paths for that file and if we don’t write the photo to disk we end up with errors we can’t figure out how to make go away. Having absolute paths specified is spaghetti code that will need to be fixed. After that you should be good to run the program. If you would like to run testing code for the backend set the TEST flag at the top of Controller.py to true and run Controller.py. If you would like to run the front end code set the TEST flag at the top of Controller.py to false and run main.py.

If you want to use the terminal:

<http://www.lowindata.com/2013/installing-scientific-python-on-mac-os-x/>

Also see the github wiki.

## Code Structure

Generally the project tries to implement a [model-view-controller](http://tomdalling.com/blog/software-design/model-view-controller-explained/) architecture. The backend team works on the model portion of the architecture, an [object-oriented](http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep) designed set of classes meant to mirror the [structures in the eye](https://lh3.ggpht.com/_j2CBHNFffn4/TD2-b5kW7lI/AAAAAAAAACc/G-j5bS4jCDk/s1600/CDR0000543553.jpg). In general each structure can detect the structures within it. For example the eye class can detect a pupil (these detections are usually done in the \_\_init\_\_ method). The controller is implemented by a module called, surprisingly, Controller.py. This provides methods that the front end team use to interface with the model.

The front end team structures their work into panels, each representing a page of the user interface. Generally the panels are named sequentially. So the first panel is panel 1, the next panel the user sees is called panel 2, etc.

Commenting conventions have been established in /doc/[DocstringExample.py](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/DocstringExample.py). Please follow these as much as is reasonable.

### Front End

See /doc/[wxPython documentation.docx](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/wxPython%20documentation.docx) to get started with wxPython and the front end code.

Take a look at the [KnownBugs.docx](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/Known%20Bugs.docx) if you’re looking for work to do.

### Backend

The Backend’s team files are all located in the /src folder of the team’s [github repository](https://github.com/UCSD-TIES/DVS-Python/). The model files are the following(from topmost in architecture to bottommost): Patient.py, FacePhoto.py, HorizontalPhoto.py,VerticalPhoto.py, and Pupil.py. Controller.py provides the interface for the front end team to interface with the backend without needing to know implementation details. You’ll notice some other files, eyeDetection.py, and eyeRemove.py. Generally in our development we’ve found it easier to write a script to perform [CV](https://en.wikipedia.org/wiki/Computer_vision) detections and then, once we have that script working, to [refactor](http://c2.com/cgi/wiki?WhatIsRefactoring) it into the Model structure. You may safely ignore these files without missing out on anything.

Controller.py’s methods are fairly simple. They just call the relevant setters and getters in the model structure and do some error checking. There’s a section at the bottom called testing that mimics calls the front end team will make to make it easier to debug the backend’s code without dealing with front end code. You may comment this out if you’re testing with the real GUI.

Now, onto the model structure, the heart of the backend. Before you proceed you will want to take a look at the [class diagram](http://msdn.microsoft.com/en-us/library/vstudio/dd409437.aspx) in the /doc folder so you can get yourself oriented. The most recent version will be named to something like UMLDiagram.png, though not exactly that. The structure is fairly simple: A Patient has a HorizontalPhoto and a VerticalPhoto both of which are FacePhotos. Each FacePhoto can detect its own Eyes. Each eye can detect its own Pupil. And a Pupil can detect its own [crescent](https://webcache.googleusercontent.com/search?q=cache:suWY3fnqp-EJ:www.medscape.com/features/slideshow/vision-screen&hl=en&gl=us&strip=1) and white dot. You’ll notice that “region” is referenced a lot in the documentation and in the comments. This is not a python or openCV type. It’s just a placeholder we’ve left there to indicate that some sort of region should be stored. This may vary. For example eye regions are stored in a rectangle format, as a tuple of four ints representing (x1,y1,x2,y2) where (x1,y1) and (x2,y2) are the opposite corners of the rectangle. (Note: OpenCV and PIL deal with rectangles differently. PIL does not use opposite corners of the rectangle but instead (x,y,w,h). (x,y) is the upper left corner of the rectangle. w is width. h is height.) Of course Pupil and sclera don’t have defined region types since the code we have now can’t detect them. We’ll probably want a circle with a center for Pupil and an some blob-like region for the sclera.

To avoid ridiculous amounts of rogue print statements in the code most modules have a module-level variable “DEBUG” at the top. If you want to see print output just set it to true and run again. If you’re putting in any print statements make sure to wrap them in an if DEBUG. Be a bit careful though, occasionally an if DEBUG will wander before an else statement, stealing it from its original parent if and messing up the control flow.

Speaking of flow, you’ll also want to take a look at the flow of execution diagram found [here](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/BackendFlowOfExecutionDiagram.PNG) to get a sense of how it all fits together. (If you are looking to update this diagram use the .zargo file in conjunction with the program ArgoUML).

Take a look at the [KnownBugs.docx](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/Known%20Bugs.docx) if you’re looking for work to do.

You may want to look at the [old source code](https://github.com/UCSD-TIES/DVS-Python/tree/master/oldSourceCode) (we’re talking like 1998 C++ code warrior here) for algorithm ideas and guidance.

### Hardware

Please refer to [HardwareDocumenation.docx](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/HardwareDocumentation.docx)

Take a look at the [KnownBugs.docx](https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/Known%20Bugs.docx) if you’re looking for work to do.

## Github 101

Our repository is [here.](https://github.com/UCSD-TIES/DVS-Python/) You’ll need to be added as a [UCSD-TIES](https://github.com/UCSD-TIES/DVS-Python/) member before you can edit the repo. You can do that by asking your team leader to add you. They’ll either have the authority to do so or know who does. [Here’s](http://rogerdudler.github.io/git-guide/) a nice tutorial on github basics. To start off clone the repository and make sure it’s working.

Some general tips and best practices:

* Make your commit messages informative
* Pull, and pull often to avoid conflicts
  + That means every time you sign on to get working with the project, even if you haven’t made any changes
* Pull before you push
* Push as often as possible so others can see your code
  + But be judicious, if your code is still being debugged and breaks others’ code keep it local or isolate it so team members can see it but it doesn’t break the build
* If you get a conflict you have to go into the file and manually edit it to make sure it’s what you want then commit that

# 

# Resources

## Documents (in the /doc folder on github)

### User Stories (not updated in Wi14)

These are short descriptions of application functionality from the user’s perspective. They’re used in the [Agile](http://agilemethodology.org/) method of software engineering and are helpful to keep track of development progress and communicate with the client.

Description of User Stories: <http://www.mountaingoatsoftware.com/topics/user-stories>

### Requirements (not updated in Wi14)

These are short descriptions of application functionality that the system needs to provide. They help narrow down user stories into actual coding/implementation strategies.

Location on github: <https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/Requirements.docx>

Description of Requirements: <http://www.microtoolsinc.com/Howsrs.php>

### Class Diagrams

This [object-oriented](http://www.codeproject.com/Articles/22769/Introduction-to-Object-Oriented-Programming-Concep) diagram lists how classes interact, outlining [inheritance and composition](http://www.artima.com/designtechniques/compoinh.html) relationships as well as listing the attributes and methods a class contains. If you get lost in the backend’s model, you should refer to this. Our diagram is written in standard UML(Unified Modeling Language) format for a class diagram.

Location on github: <https://github.com/UCSD-TIES/DVS-Python/blob/master/doc/UML_Diagram_DVS.png>

Description of class diagram: <http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/>

## Links/Quick Reference

### Technologies

[Python Docs](http://docs.python.org/2.7/)

[OpenCV Docs](http://docs.opencv.org/index.html) – Most of the docs are for C++ so generally you’ll have to ctrl+f for the python related parts or do some googling

[Short python/opencv talk](http://pyvideo.org/video/1796/simplecv-computer-vision-using-python)

[PIL Docs](http://effbot.org/imagingbook/pil-index.htm)

[wxPython Docs](http://wiki.wxpython.org/)

### Editors

[IDLE](http://www.ai.uga.edu/mc/idle/)

[Vim](http://www.vim.org/download.php)

[PyDev](http://pydev.org/download.html)

[Sublime](http://www.sublimetext.com/download) (Recommend)

[Canopy](https://www.enthought.com/products/canopy/) (Recommend)

### Git

[UCSD-TIES Organization Page](https://github.com/UCSD-TIES/)

[Our repository](https://github.com/UCSD-TIES/DVS-Python)

[Explanation of Git](http://rogerdudler.github.io/git-guide/)

[Overview of Git Commands](http://gitref.org/)

### Eye Detection

[Source of the algorithm we’re currently using](http://japskua.wordpress.com/2010/08/04/detecting-eyes-with-python-opencv/)

### Pupil and Iris Detection

[Source of the algorithm we’re currently using](http://opencv-code.com/tutorials/pupil-detection-from-an-eye-image/)

### Sclera/Crescent Detection

[Color detection in openCV](http://webcache.googleusercontent.com/search?q=cache:o30esb4Aq4QJ:www.aishack.in/2010/07/tracking-colored-objects-in-opencv/+&cd=1&hl=en&ct=clnk&gl=us)

### Software Concepts

[Model View Controller (MVC)](http://tomdalling.com/blog/software-design/model-view-controller-explained/)

[Agile](http://agilemethodology.org/)

[Scrum](http://www.youtube.com/watch?v=XU0llRltyFM)

[Refactoring](http://c2.com/cgi/wiki?WhatIsRefactoring)

[User Stories](http://www.mountaingoatsoftware.com/topics/user-stories)

[Requirements](http://www.microtoolsinc.com/Howsrs.php)

[Class Diagram](http://www.ibm.com/developerworks/rational/library/content/RationalEdge/sep04/bell/)

### Miscellaneous

[Eye Structures](http://www.eyesightresearch.org/background.htm)

[Sclera](https://en.wikipedia.org/wiki/Sclera)

[Concepts behind the project(.ppt)](https://www.dropbox.com/sh/8nsmamaiklpvr9l/CDAFMoZM7A/Dr.%20Bartsch%27s%20DVS%20slides/vision-screening.ppt)

ROI – Stands for Region of Interest. It represents the portion of the photo that OpenCV will be working on when you give it commands. You can set and reset this manually.