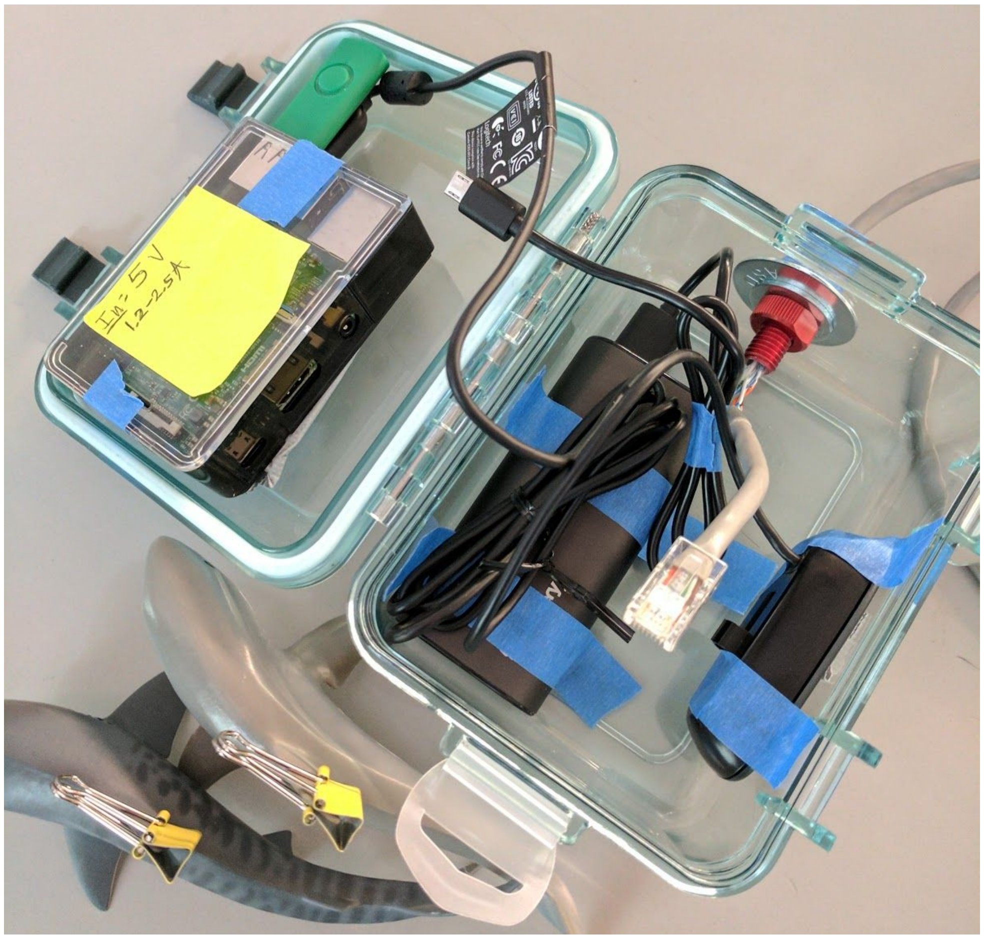
Underwater Computer Vision - Fish Recognition



Prepared by
Spencer Chang - spencerd56@gmail.com
Austin Otto - austinreedotto@gmail.com

Advised by
Andrew Danowitz - adanowit@calpoly.edu

CPE Senior Project

June 16, 2017

Table of Contents

Introduction

The Underwater Computer Vision – Fish Recognition project includes the design and implementation of a device that can withstand staying underwater for a duration of time, take pictures of underwater creatures, such as fish, and be able to identify certain fish. The system is meant to be cheap to create, yet still able to process the images it takes and identify the objects in the pictures with some accuracy. The device can output its results to another device or an end user.

System Requirements

The system…

- Shall operate underwater without damage to its components.
- Shall stay underwater for a long period of time.
- Shall take clear pictures of objects and fish while underwater.
- Shall run image processing and machine learning on images without outside help.
- Shall be able to run for a long period of time with a battery or external power source.
- Shall have a way to output results of computations to outside devices while underwater.

System Specifications

Weight	3 lb.
Dimensions	6.625” x 5.125” x 4.125”
Image Resolution	640x480
Battery Life	6000 mAh

System Architecture

The system is contained within a waterproof GSI Outdoors Lexan Gear box containing a Raspberry Pi 3, Jackery Portable Charger, and Logitech C310 webcam. The Raspberry Pi is powered by the portable charger and takes images through the C310. The images taken by the webcam are stored on a 4GB flash drive, and results from the computation will be sent out using an Ethernet cable put through a cable penetrator to an outside device. The outside device would communicate with the Raspberry Pi 3.

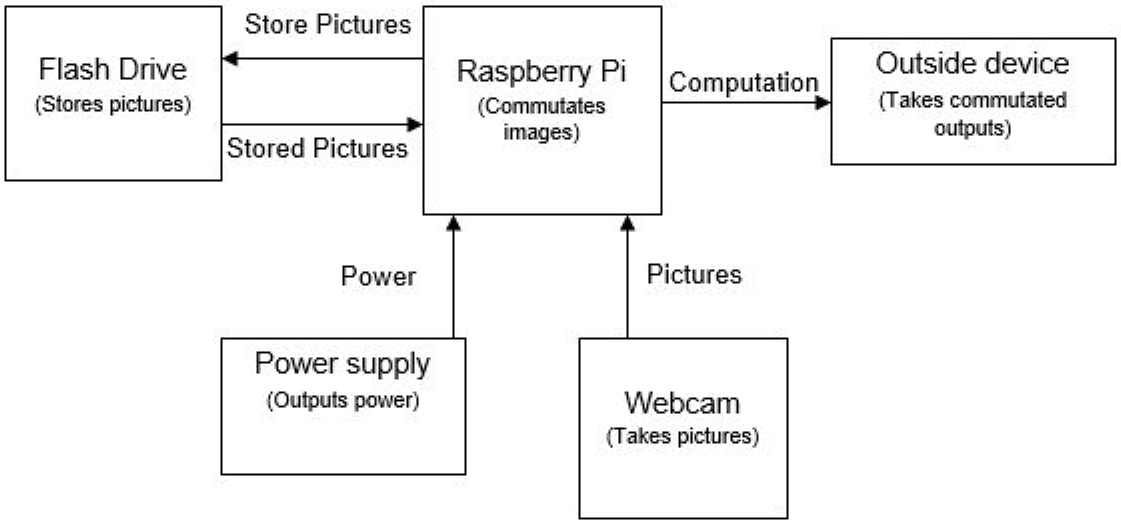


Figure 1: System Block Design

Component Design

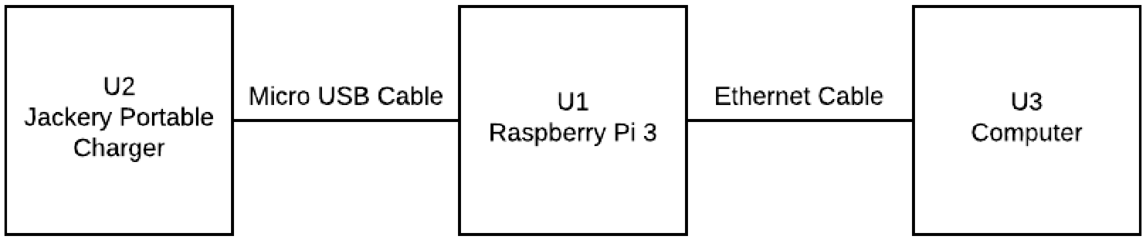


Figure 2: Schematic Diagram

Bill of Materials

	Part	Part #	Distributor	Supplier Item #	Quantity	Unit Price	Extended Price
1	Raspberry Pi 3	DEV-13825	Sparkfun	DEV-13825	1	$39.95	$0
2	Straight-Through Ethernet Cable	(Unknown)	EE Tech Support	(Unknown)	1	???	$0
3	Cable Penetrator	PENETRATOR-10-25-A-R2	BlueRobotics	PENETRATOR-10-25-A-R2	1	$4.00	$4.00
4	Logitech C310	960-000585	Amazon.com	B003LVZO8S	1	$15.00	$15.00
5	Jackery 6800mAh Portable Charger	(Unknown)	Amazon.com	(No Longer Sold by Supplier)	1	$14.99	$14.99
6	Toy Fish	(Unknown)	Tom's Toys	(Unknown)	2	$7.99	$14.98
7	GSI Outdoors Lexan Gear Box	735-Lexan	Amazon.com	B004P8BIPE	1	$13.60	$13.60
						Total	$62.57

Note: The Raspberry Pi 3 and Ethernet cable were provided to the group. Therefore, no costs were incurred.

System Integration

Hardware Integration

To prevent the Raspberry Pi, power source, and camera from getting damaged by being underwater, a small waterproof container contains all of the equipment that was meant to be the main device. At first, an external device was going to use Wi-Fi to communicate directly with the system. However, Wi-Fi does not work underwater. The Raspberry Pi's Ethernet was used for communication instead. A hole was drilled into the side of the GSI box and a cable penetrator was inserted to allow an Ethernet cable to reach the Raspberry Pi and allow an external device to communicate with the system.

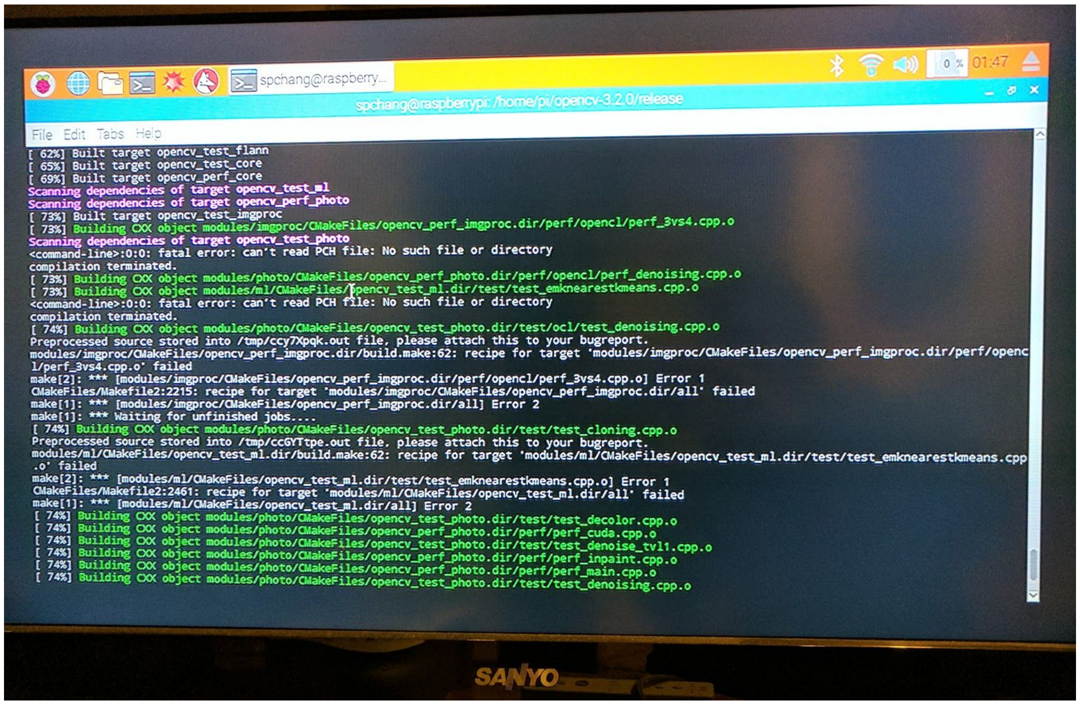
The device is meant to sink underwater without additional assistance, but its buoyancy does not allow it to do so. Therefore, a weight or device has to be used to hold the system underwater.

Software Integration

Following the instructions given through the Raspberry Pi site [1], Raspbian was installed on the provided 8GB SD card. Additional libraries for different desktop environments were installed later.

OpenCV 3.2.0 took the longest to install in terms of Raspbian software integration. By following the PyImageSearch tutorial [2], OpenCV installation started out fairly smooth. Certain OpenCV dependencies had to be searched for online or through a 'sudo apt-cache search' command as the libraries may not necessarily have the same values as those mentioned in the tutorial. An error occurred just after reaching 70% completion, where it was claimed that the Makefile could not find a certain "PCH file" (Figure 3). After re-trying the 'make install' command to install OpenCV 3.2.0 on Raspbian multiple times using all 4 cores of the Raspberry Pi, it was reasoned that race conditions were at fault for causing such problems. After toning down to 2 cores, the OpenCV installation was successful, almost instantaneously.

For better object recognition, SIFT/SURF will be useful when installed. Both are 'opencv-contrib' addons to OpenCV. The steps for installing these would be to uninstall openCV-3.2.0 or delete its installed directory, get the virtualenv and virtualenvwrapper as mentioned in the tutorial [2], and cmake/install openCV-3.2.0 with openCV_contrib-3.2.0 in a virtual environment called 'cv'. All of this is recommended by the tutorial above. Virtual environments are recommended since they isolate any problems that may occur during programming. It is a great way of testing things without the possibility of crashing the system.



```
[ 62%] Built target opencv_test_flann
[ 65%] Built target opencv_test_core
[ 69%] Built target opencv_perf_core
Scanning dependencies of target opencv_test_ml
Scanning dependencies of target opencv_perf_photo
[ 73%] Built target opencv_test_imgproc
[ 73%] Building CXX object modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/perf/opencl/perf_3vs4.cpp.o
Scanning dependencies of target opencv_test_photo
<command-line>:0:0: fatal error: can't read PCH file: No such file or directory
compilation terminated.
[ 73%] Building CXX object modules/photo/CMakeFiles/opencv_perf_photo.dir/perf/opencl/perf_denoising.cpp.o
[ 73%] Building CXX object modules/ml/CMakeFiles/opencv_test_ml.dir/test/test_emknearestkmeans.cpp.o
<command-line>:0:0: fatal error: can't read PCH file: No such file or directory
compilation terminated.
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_test_photo.dir/test/ocl/test_denoising.cpp.o
Preprocessed source stored into /tmp/ccy7Xpqk.out file, please attach this to your bugreport.
modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/build.make:62: recipe for target 'modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/perf/openc
l/perf_3vs4.cpp.o' failed
make[2]: *** [modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/perf/opencl/perf_3vs4.cpp.o] Error 1
CMakeFiles/Makefile2:2215: recipe for target 'modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/all' failed
make[1]: *** [modules/imgproc/CMakeFiles/opencv_perf_imgproc.dir/all] Error 2
make[1]: *** Waiting for unfinished jobs....
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_test_photo.dir/test/test_cloning.cpp.o
Preprocessed source stored into /tmp/ccGYTtpe.out file, please attach this to your bugreport.
modules/ml/CMakeFiles/opencv_test_ml.dir/build.make:62: recipe for target 'modules/ml/CMakeFiles/opencv_test_ml.dir/test/test_emknearestkmeans.cpp
.o' failed
make[2]: *** [modules/ml/CMakeFiles/opencv_test_ml.dir/test/test_emknearestkmeans.cpp.o] Error 1
CMakeFiles/Makefile2:2461: recipe for target 'modules/ml/CMakeFiles/opencv_test_ml.dir/all' failed
make[1]: *** [modules/ml/CMakeFiles/opencv_test_ml.dir/all] Error 2
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_test_photo.dir/test/test_decolor.cpp.o
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_perf_photo.dir/perf/perf_cuda.cpp.o
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_test_photo.dir/test/test_denoise_tvl1.cpp.o
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_perf_photo.dir/perf/perf_inpaint.cpp.o
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_perf_photo.dir/perf/perf_main.cpp.o
[ 74%] Building CXX object modules/photo/CMakeFiles/opencv_test_photo.dir/test/test_denoising.cpp.o
```

Figure 3: OpenCV 'make install' PCH File Error

VNC Viewer was installed on both the Raspberry Pi [3] and the PC used during the development of this project. VNC is nearly synonymous with the Remote Frame Buffer (RFB) protocol. On a high-level, the Raspberry Pi and PC communicate with a server-client relationship. When connecting, the Raspberry Pi (server) challenges the PC (client) to respond with a valid username and password. After valid credentials are sent to the Raspberry Pi, both devices communicate window size, frame buffers, and other things for their protocol. Once those are set, the PC can now communicate with the Raspberry Pi.

The final software implementation done to Raspbian was the installation of TensorFlow 1.1. Since Raspbian is a Debian Linux installation, the following instructions may be followed to install TensorFlow on the Raspberry Pi [4]. No major, lasting bugs were present in the installation. On one incidence, TensorFlow output minor errors that went away when a clean of the files was done and installation restarted. A simple TensorFlow program is as below. It is considered the "Hello, World!" of TensorFlow.

```python
import tensorflow as tf
hello = tf.constant('Hello, TensorFlow!')
sess = tf.Session()
print(sess.run(hello))
# -> Hello, TensorFlow!
```

Cameras

Multiple cameras were considered for use in this project, including Cognex, SeaViewer, GoPro, and Logitech. Before the idea of placing the entire system into an underwater box, underwater cameras were considered for use, but many were too expensive for our system requirements. Since the Raspberry Pi 3 was in use, a webcam was seen as a balance of low cost and decent image quality.

Within the family of Logitech webcams, two different products were tested to see which would provide the best image quality. The C210 and C310 were tested, and the images taken by the cameras were compared by the team. Both held passable quality for images, but it was found that the C310 had better white balance. The C310 was connected to the Raspberry Pi and, using bash scripts, could take pictures and store them on a 4GB flash drive. See Appendix C for all bash scripts.

To bring everything under a single program, the ‘fswebcam’ command was integrated into a Python 3 program using subprocesses shown below.

```
homeDir = "/home/pi/";
destDir = "/mnt/usb/uwphotos/";

outImg = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S");
outImg = "test_" + outImg + ".jpg";
camIn = ['fswebcam', '-r', '640x480', '-d', '/dev/video0', '-i', '0', outImg];
mvPic = ['mv', (homeDir + outImg), (destDir + outImg)];
print("TAKING IMAGE", (count + 1), "- " + outImg);
print("*****Start*****");

# Call 'fswebcam' to take the image
process = Popen(camIn, stdout=PIPE, stderr=PIPE);
stdout, stderr = process.communicate();
print("---STDERR---");
print(stderr.decode("utf-8"));
time.sleep(5);

# Call 'mv' to move the image to the USB drive
process = Popen(mvPic, stdout=PIPE, stderr=PIPE);
stdout, stderr = process.communicate();
```

Errors occasionally appear when images were taken in quick succession. Raspbian would say the “device or resource” is busy or that the “image palette could not be used”, perhaps referring to

RGB. The code for taking pictures was then altered to have a slight delay between image captures to make the errors occur less often.

Overall, the majority of work was spent creating the hardware, leaving little room for learning OpenCV and TensorFlow in time to get an adequate machine learning algorithm implemented. Currently, code was written to utilize the built-in OpenCV GrabCut algorithm. Smaller tests were also done to use color ranges [5] to segment the background out of the picture. For instance, pixels with RGB values between `[17, 15, 100]` and `[50, 56, 200]` would be considered red. Pixels outside of this range are considered the background. Note that OpenCV reads pixel values as BGR. These image segmentation attempts were ultimately unsuccessful, but the tools for recognition of objects are installed.

Project Outcomes and Deliverables

Completed

The system is able to run both OpenCV and TensorFlow without any external help. There is not much progress in processing images to segment an object from its background. Attempts were made to use the built-in OpenCV GrabCut algorithm, but the background was still seen as part of objects as seen in an image comparison below.

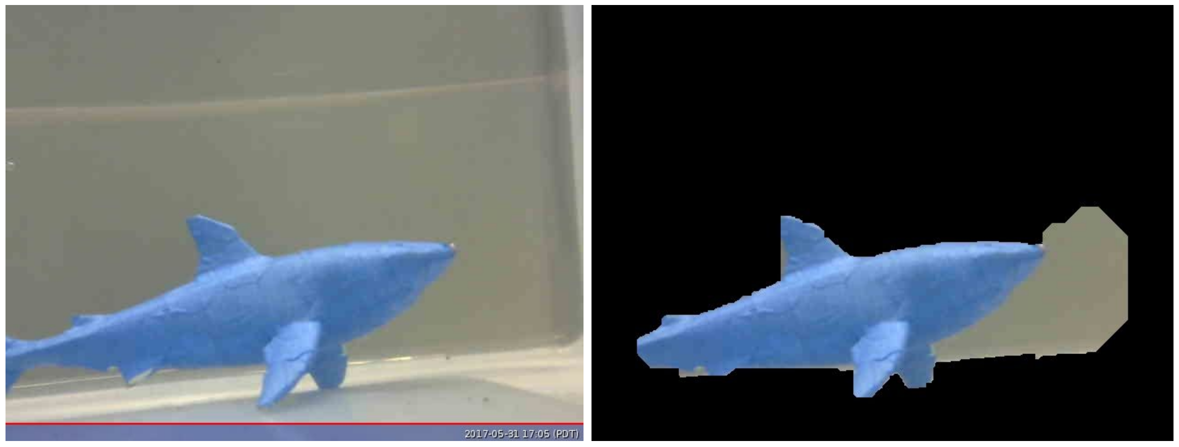


Figure 4: Original and Masked Image of Underwater Toy Fish

With VNC Viewer installed on the Raspbian OS and any other external device, communication with and control of the Raspberry Pi is easy. It was found that the Raspberry Pi 3 draws a low amount of energy when taking pictures, but a supposition was stated that the power consumption would greatly increase with the usage of TensorFlow's neural networks and OpenCV's built-in modules.

Future Work

Two major improvements standout for the system. The first is to include heavier weights within the GSI box. A recommendation would be to look into fisherman's bank sinkers, which are dense and heavy weights commonly used within the fishing community. The trade-off with adding more weight is between insulation of heat and weight of the box. With more things inside the apparatus, the heat generated from the components will have less airflow. As discussed, the benefit of adding weights is that the box will sink by itself. The second is to implement a better image processing using OpenCV and a neural network algorithm on the Raspbian OS that takes all images and trains the algorithm to recognize the correct objects. This is the next and hardest step in development.

Conclusion

Overall, this was a difficult project to complete. Ramping up on machine learning proved to be the hardest part. TensorFlow is an advanced library within machine learning and should be tackled by members of a group with adequate time available. Good image processing continues to hold a high learning curve. *If the focus of a project is image processing and machine learning, make the picture-taking apparatus quickly and efficiently, so a testing platform can be created for software implementation and testing.* Time was wasted in creating the underwater box for this project, a mistake that appeared benign but came back to undermine any progress made. In the future, spend most time working on the TensorFlow and machine learning portion of the project instead of on the hardware necessary to create the underwater apparatus.

References

[1] Raspberry Pi Foundation. (2017). *Raspbian* [Online]. Available: https://www.raspberrypi.org/downloads/raspbian/.

[2] Rosebrock, Adrian. (2017, April 16). *Install guide: Raspberry Pi 3 + Raspbian Jessie + OpenCV 3 (1st)*. [Online]. Available: http://www.pyimagesearch.com/2016/04/18/install-guide-raspberry-pi-3-raspbian-jessie-opencv-3/.

[3] Raspberry Pi Foundation. (2017). *VNC (Virtual Network Computing)*. [Online]. Available: https://www.raspberrypi.org/documentation/remote-access/vnc/.

[4] Google. (2017, April 26). *Installing TensorFlow on Linux*. [Online]. Available: https://www.tensorflow.org/install/install_linux.

[5] Rosebrock, Adrian. (2014, August 4). *OpenCV and Python Color Detection*. [Online]. Available: http://www.pyimagesearch.com/2014/08/04/opencv-python-color-detection/

Appendix A - User's Manual

Connect and Power On the System

1. Connect the USB webcam to the lower right USB port on the Raspberry Pi.
2. Connect a USB flash drive to the top right USB port on the Raspberry Pi. The port is chosen for the device driver '/dev/sda1' seen in the mounting and unmounting bash scripts below.
3. Connect the Jackery 6000mAh Portable Charger to the Raspberry Pi's microUSB port.
4. Plug the Ethernet cable that has been threaded through the box into the Raspberry Pi's Ethernet port and the other side to the desired computer.
5. Turn on the device by turning on the portable charger.
6. Once Raspbian boots up, connect using VNC Viewer and enter valid credentials when prompted.
7. Open the command line and 'su', or set user, as someone included in the sudoers file. This allows you to use 'sudo' commands for mounting, unmounting, taking pictures, and moving files.
8. Run the mount.sh bash script with sudo privileges to mount the USB drive onto the Raspbian filesystem. Any program referring to '/mnt/usb' can now write/read files without knowing the exact USB-specific name.

Mouse (if working on Pi w/o VNC)	USB Flash Drive
Keyboard (if working on Pi w/o VNC)	Webcam

Table 1: USB Port Layout

Take Pictures

Prerequisite: System is correctly connected and powered on.

1. Run the Python 3 webcam code with 'python3 <filename>'. Since Python 2 and 3 are both installed on the Raspberry Pi, 'python3' chooses Python 3 over 2.
2. After a few seconds the program will take a picture with the name format "test-<year>-<month>-<day>_<hour>-<minute>-<second>.jpg" and place it in the directory "/mnt/usb/uwphotos/". The program has a 5-second wait after each image.
3. The code will repeat step 2 four more times before exiting. The number of pictures taken can be changed with the limit variable.

Use grabcut.py

Prerequisite: System is correctly connected and powered on:

1. Access the Raspbian console and enter: “python3 grabcut.py” while in the same directory as grabcut.py.
2. The program will ask for an image. Make sure the image is in the same directory as grabcut.py. This is due to the way grabcut.py is currently programmed.
3. Enter the image’s entire name.
4. After a short period of time it will finish its process and produce an image called “masked-<image’s name>” and place it in the current directory. The program has a set of hardcoded coordinates that tell the GrabCut algorithm where to check for foreground objects. This may be changed using any kind of text editor.
5. Repeat steps 2-4 or enter “end” to stop the process.

Appendix B - Code

B.1 - Original ‘fswebcam’ Bash Script, or “webcam.sh”

```
#!/bin/bash

if [ $# -eq "2" ]; then
   DATE=$(date +'%Y-%m-%d_%H-%M-%S');
   NAME="$1_$DATE";
   sudo fswebcam -r 640x480 -D $2 -d /dev/video0 -i 0 $NAME.jpg;
   sudo mv $NAME.jpg /mnt/usb/uwphotos/$NAME.jpg;
else
   echo "Usage: ./webcam.sh pictureName delay(sec)";
fi
```

B.2 - Mounting

```
sudo mount -v /dev/sda1 /mnt/usb
```

B.3 - Unmounting

```
sudo umount -v /dev/sda1
```

B.4 - Webcam Python 3 Code

```
import subprocess
import datetime
import time
from subprocess import Popen, PIPE

homeDir = "/home/pi/";
```

```
destDir = "/mnt/usb/uwphotos/";

count = 0;
limit = 5;
while (count < limit):
   outImg = datetime.datetime.now().strftime("%Y-%m-%d_%H-%M-%S");
   outImg = "test_" + outImg + ".jpg";
   camIn = ['fswebcam', '-r', '640x480', '-d', '/dev/video0', '-i', '0', outImg];
   mvPic = ['mv', (homeDir + outImg), (destDir + outImg)];
   print("TAKING IMAGE", (count + 1), "- " + outImg);
   print("*****Start*****");

   # Call 'fswebcam' to take the image
   process = Popen(camIn, stdout=PIPE, stderr=PIPE);
   stdout, stderr = process.communicate();
   print("---STDERR---");
   print(stderr.decode("utf-8"));
   time.sleep(5);

   # Call 'mv' to move the image to the USB drive
   process = Popen(mvPic, stdout=PIPE, stderr=PIPE);
   stdout, stderr = process.communicate();
   print("---STDERR---");
   print(stderr.decode("utf-8"));
   print("*****End*****");

   count += 1;
   time.sleep(5);
```

<u>B.5 - grabcut.py</u>

```
#image Segmentation
import numpy as np
import cv2

x = 'test'

while (x != 'end'):
   x =  input('Image: ')
   if(x != 'end'):
      img = cv2.imread(x)
      mask = np.zeros(img.shape[:2],np.uint8)

      bgdModel = np.zeros((1,65),np.float64)
      fgdModel = np.zeros((1,65),np.float64)

      rect = (50,50,590,400)
      cv2.grabCut(img,mask,rect,bgdModel,fgdModel,5,cv2.GC_INIT_WITH_RECT)
```

```
mask2 = np.where((mask==2)|(mask==0),0,1).astype('uint8')
img = img*mask2[:,:,np.newaxis]
cv2.imwrite('masked-' + x, img)
```