# OpenIVIS Summer 2024 Report

Hailey Knolton Ruby Peterman

2024-07-11

# Table of contents

# 1 OpenIVIS Summer 2024 Quarto Report

# Table of Contents

# 2 Abstract

This paper presents the continuation of the design of an open source, inexpensive, and modular In Vivo Imaging System (IVIS) from the Harvey Mudd Biophotonics Lab during the summer of 2024. The authors contributions include the implementation of fluorescence imaging and laser speckle imaging on an improved physical system design. The system was created using widely available materials and tools including acrylic, a laser cutter, a 3D printer, and a Raspberry Pi computer. The experiments done to verify fluorescence imaging resulted in the expected relationship between fluorescence and fluorescent dye concentrations with a set exposure time of 0.1 seconds. The experiments done to implement and verify laser speckle imaging have assisted in the development of its abilities as an IVIS imaging method, but need further work to provide accurate results. The results from the experiments demonstrate the improved capabilities of the new IVIS system and pathways for future developments.

# 3 Introduction

## 3.1 In Vivo Imaging Systems

In Vivo Imaging Systems (IVIS) are optical imaging devices used in scientific research to create 2D and 3D representations of biological organisms and processes non-invasively. These systems use advanced imaging techniques, such as bioluminescence and fluorescence imaging, to visualize and track various biological activities within an organism over time. This approach to optical imaging can assist in drug development, understanding disease behaviors, or other biological processes in their natural context [Refaat]. Most IVIS Imaging methods implement optics in order to extract information about a subject based on the optical properties of the subject and the technique being applied. Many commercial IVIS systems can provide additional capabilities such as X-Ray, temperature control, computed tomography (CT), or accessories [KU, Revvity]

While IVIS systems are able to create detailed images over a broad range of applications, they can be limiting due to their inaccessibility. IVIS systems tend to cost upwards of $100,000, which may bar smaller or less-funded research institutions from purchasing their own system [bostonind]. Renting the use of an IVIS system is typically in the hundreds of dollars range as well, which further restricts the accessibility of in vivo imaging [OSU]. In addition to being high-cost, commercial IVIS systems are also restricted to the imaging applications they've been developed for with little room for customization or modularity [Source?].

The goal of the OpenIVIS project is to create a low-cost, open source and modular version of an IVIS system for biological imaging. Free open-source software (FOSS), free open-source hardware (FOSH) and the increased accessibility of rapid prototyping techniques, such as 3-dimensional (3D) printing, would allow for any institution to implement a version of this system in their research. An IVIS system with a modular design would also permit users to replace the imaging techniques used in order to best advance their work. Additionally, this would pave the way for implementation of in-vivo imaging techniques not currently available in most commercial systems such as Laser Speckle Contrast Imaging (LSCI). SHOULD maybe reference previous work on this, especially the CSM/HMC paper.

## 3.2 Fluorescence Imaging

One of the most common capabilities of IVIS systems is fluorescence imaging. Fluorescence is a highly sensitive analytical tool that is used to measure extremely low concentrations of a compound in a solution [1]. The Jablonski diagram shown in Fig. # depicts the fluorescent process. When light is absorbed by a compound, molecules of that compound will become excited and raise to a higher energy level. Fluorescent compounds usually contain conjugated double bonds, where a certain number of electrons have greater mobility than the other electrons in the molecule [1]. This greater mobility allows for more molecules to become excited when the light is absorbed. When these molecules return to their ground state, some of the energy is emitted as fluorescence.

The energy that makes up light are called photons. Photons that absorb and excite molecules hold a certain amount of energy that determines their wavelength, or color [2]. When a molecule emits a photon as it returns to ground state, the energy in the photon that is emitted is less than in the photon that was excited. This means that the resulting photon will have a longer wavelength and a different color [2].

In order to image fluorescence, the absorbed and emitted photons of light must be controlled based on their wavelength spectrums. The excitation wavelength spectrum and the emission wavelength spectrum can often overlap, allowing the camera to capture photons of both wavelengths. An example of these spectrums is shown in Fig. #. In order to see the fluorescence of a compound, only the emitted light must be captured by the camera. Implementing an optical filter can help to control what wavelengths are captured. Optical filters allow for wavelengths of a certain range to be the only wavelengths detected by a camera by filtering out other wavelengths Fluorescence imaging has a variety of applications including medical imaging, environmental monitoring, and biological research. A common application of fluorescence is to non-invasively analyze biological molecules in vivo. Most IVIS systems use fluorescence for this purpose, and OpenIVIS will also demonstrate this fluorescence capability.

## 3.3 Laser Speckle Contrast Imaging

Laser Speckle Contrast Imaging (LSCI) is an optical imaging technique used to track movement, such as blood flow, by visualizing blur. When a diffuse object is illuminated with coherent light, it produces scattered light waves which can be visualized as a random interference pattern called a speckle image [Boas]. In order to determine the size of the speckles in the speckle pattern, autocorrelation can be applied to the image. Autocorrelation compares the intensity of the speckle pattern at two different points by multiplying values across the entire image. The autocorrelation can also be found by taking the fourier transform of the images intensity distribution [Wikipedia]. Eq. 1 shows a speckle pattern's autocorrelation calculation by taking the Fast Fourier Transform of the image's intensity I(x,y). The transform is then multiplied by its complex conjugate, noted by the asterisk, in the fourier domain to obtain

Fcc in Eq. 2. Finally, the inverse fourier transform returns the calculation to the spatial domain, resulting in the autocorrelation FA as shown in Eq. 3. Given the autocorrelation of a speckle image, the size of the speckle in pixels can be determined by finding the width of the autcorrelation's peak at half of its maximum, often referred to as the full width half max (FWHM). Most speckle images have a speckle size of one to two pixels.

$$F_{fft}(f) = \text{FFT}\{I(x,y)\} \quad \text{(Eq.1)} \tag{3.1}$$

$$F_{cc}(f) = F_{fft}(f) \times F^{*}_{fft}(f) \quad \text{(Eq.2)} \tag{3.2}$$

$$F_{A}(t) = \text{IFFT}\{F_{cc}(f)\} \quad \text{(Eq.3)} \tag{3.3}$$

Static speckle images have high contrast patterns but when movement is imaged, the fluctuations in intensity can cause the contrast between neighboring speckles to decrease. The speckle contrast, K, can be derived as the standard deviation of pixel intensity over the mean pixel intensity, as shown in Eq. 4. Moving objects, such as blood flowing in a vein, causes the speckle pattern to shift, or decorrelate [Briers]. When this occurs, the intensity of neighboring speckles will become more similar, decreasing their contrast value.

$$K = \frac{\sigma}{< I >} \quad \text{(Eq.4)} \tag{3.4}$$

In order to compute the contrast of a full speckle image, a small window is applied to the original speckle pattern, typically 5x5 or 7x7 pixels large. This window, often referred to as a "neighborhood" is run over the entire image, computing the speckle contrast K for the intensities at each location before shifting over by 1 pixel at a time. The image is then reconstructed using the respective K values in order to produce a laser speckle contrast image. A speckle image is shown in Fig. #a, and its corresponding LSCI reproduction is shown in Fig. #b. By comparing the contrast patterns between different speckle images over time, the velocity of the movement being imaged can be determined.

# 4 Methods

## 4.1 Physical Construction

The OpenIVIS box is made out of laser cut acrylic and 3D printed parts. Black acrylic was cut to form the base, back, and sides of the box. The acrylic pieces have extruding parts along their rims to connect in a jigsaw puzzle configuration. This ensures that no light can leak out of the box and that the box is stable. The front of the box is made from a 3D printed PLA filament and has long notches to allow for the black acrylic door to slide up and down. Fig. # shows the fully assembled box.

The box has two replaceable lids for the different imaging methods.The first lid is designed for laser speckle contrast imaging. The lid is made out of black acrylic with three holes cut into it. One hole holds the camera while the other two holes hold the laser diode configuration described in Section 2.2. Fig. # shows the assembly used for LSCI.

The second lid is designed for fluorescence imaging. It has an upper layer of black acrylic jigsaw pieces. On the underside of the lid is a 3D printed ring to hold an array of Neopixel LEDs. Beneath the LED array is a diffusion plate made of white acrylic to allow for the LEDs to shine through and into the box. A 3D printed camera holder is secured to the white acrylic and has space for a 3D printed optical filter holder to slide in beneath the camera. Fig. # shows the fully assembled fluorescence top. This configuration can also be used for other imaging methods that implement the Neopixel LED array, as discussed in Section 2.3.4.

## 4.2 Electrical Components and Software

The OpenIVIS system runs primarily off of a Raspberry Pi 4, and other cheap, off-the-shelf electronics. The Raspberry Pi is a series of small single-board computers developed to be an educational tool, which has many interfacing options such as general purpose input/output pins (GPIO) that can connect to sensors. The camera used for all images in this project is an Arducam IMX519 color camera with a 24” flex cable, which is Raspberry Pi compatible and has autofocus features. A desktop and mouse are connected to the USB interfaces of the Raspberry Pi in order to use the system and make any necessary alterations. The system is stored on a 128 GB microSD card with the appropriate Raspberry Pi operating system.

For most imaging methods used by the OpenIVIS system, an LED array of Adafruit Neopixel RGB LED modules is used to illuminate subjects under study. In order to connect to the Raspberry Pi, a Raspberry Pi prototyping board is wired along with a logic level converter and terminal block as shown in Fig. #, as well as powered by an external 5V power supply. The LED array is connected to the pulse width modulated (PWM) pin from the Raspberry Pi to provide control. The LED's are affixed to the 3D printed LED array as described in section 2.1. Unmounted, 25 mm thick long-pass optical glass filters were sourced from Thorlabs in cutoff wavelengths of 515 nm, 570 nm, 665 nm and 695 nm.

LSCI and other laser based imaging techniques implement a red ThorLabs 635 nm Collimated Laser Diode Module, along with Thorlabs' RA90, SM1TC, SM1D12D, GBE05-A, LDS5, TR3, TR6, CPS635R, SM1S10, SM1T1 and AD11F to properly configure the laser beam. The laser configuration as shown in Fig. #a is affixed to the lid of the OpenIVIS box using a 3/8" 1/4-20 bolt, as shown in Fig. #b.

The majority of the code used to run the OpenIVIS system is written in the programming language Python, and available in the project's public GitHub repository [Git]. Descriptions of the Python scripts necessary to collect and process data are described in the repository's "Read Me" file. The list of necessary Python Packages is listed in Appendix #, and is run on the Raspberry Pi through a virtual environment. The Neopixel LED array and Arducam camera Python libraries are also used and require installation. Additional data processing code was written using Mathworks' MatLab software.

A full list of materials, diagrams, and relevant code is available in Appendix #, Appendix # and Appendix #, respectively.

Figs to make: LED wiring, Laser configuration, + Appendix BOM and any extra diagrams?

## 4.3 Experiments

### 4.3.1 Verification

Before fluorescence and laser speckle contrast imaging, it was necessary to verify that the Arducam camera and the LED lights were working properly. The Neopixel Python library allows for brightness control in two forms. The first is through changing the brightness level of the LED output, which can be set to any value between 0 and 1 arbitrary units. In addition to brightness level, the usage of 4 valued color codes allow for the user to change both the color of the LED output and the brightness of each color. The red, green, blue and white brightness values range from 0 to 255 arbitrary units.

Brightness level and exposure time verification can be done by changing the exposure time of the camera while keeping the brightness constant and by changing the brightness level of the LEDs while keeping the exposure time constant. The exposure time, or shutter speed, controls how long the camera is exposed to light when taking an image, and it is controlled

using picamera2 settings. The brightness level of the LEDs is controlled using the NeoPixel library settings. Images were taken at full brightness and varying exposure times of 0.001, 0.005, 0.01, 0.05, and 0.1 seconds. Images were also taken at exposure times of 0.01 and 0.1 seconds with varying brightness levels from 0 to 1 arbitrary units. These settings for these images were then compared to the intensities of the pixels in the images.

The pixel intensities were measured by taking a three by three set of pixels near the center of the image and putting them in an excel spreadsheet. To plot the data a MATLAB script was used to read in the intensity values and average across each set of nine. These averaged values were then averaged across five different trials of varying the brightness level and exposure time. The expected results are positive trends between exposure time and pixel intensity and brightness level and pixel intensity.

Since the above process will only verify the LEDs based on what the camera is detecting from the LED light, it was also necessary to test the actual LED output using a photodiode. The photodiode outputs wattage values based on the intensity of light detected. Changing both the brightness levels from the NeoPixel settings and the color code values of the red, green, blue, and white LEDs should result in positive trends between brightness value and output wattage. The photodiode was placed inside of the system's main box compartment with the sensor facing upward. Data was collected for three different sensor locations in the box. For each location, a Python script was used to cycle through the Neopixel brightness levels from 0 to 1, increasing by 0.1 each time to achieve 11 equally distributed brightness levels. For each brightness level, the color code tuples were also cycled from 0 to 255 for each LED color. The color code values increased by 26 a.u. each cycle to achieve 11 equality distributed values. The LED output in nW was saved in a spreadsheet for each setting, resulting in XXX values for each photodiode sensor location. The trends in the white LED output were analyzed for changes in brightness level at a constant color code of 255 a.u., and the changes in color code for a constant brightness level of 1. For each setting, the average wattage and standard deviation was taken across the three trials and inputted into a MatLab script that created graphs of those values across changes in Neopixel setting.

### 4.3.2 Fluorescein Ssothiocyanate

To test the fluorescence capabilities of OpenIVIS, ten concentrations of the commonly used fluorescent dye, Fluorescein isothiocyanate (FITC), were diluted in ethanol. FITC emits fluorescence of wavelengths ranging in the green spectrum with a peak around 530 nm [3]. To excite the FITC solutions, blue LEDs were used to illuminate the dye at full brightness. The Neopixel blue spectrum has a peak approximately 450 nm. To prevent the camera from detecting the excitation light, a 515 nm long pass optical filter was placed in front of the camera. The filter blocked out all wavelengths lower than 515 nm, including the blue excitation light. The ten different concentrations of FITC ranged from 10-3 M to 10-12 M. In a 96-well PCR plate, 2 wells of each FITC concentration were placed in a four by five orientation and put into the OpenIVIS box for imaging.

To quantify the fluorescence of each FITC concentration, the raw image data from the captured images was converted into Bayer images. The Bayer images have a red, green, green, blue (RGGB) color filter array as shown in Fig. #. In this color filter array there is only one color component in each pixel of the image [4]. The other two color components for the pixel must be interpolated from information gathered from its neighboring pixels [4]. After getting the Bayer images, the first green channel was isolated so that the images have empty and filled values ranging from 0 to 1023 arbitrary units. These images allow for the intensity of the pixels to be extracted and compared across multiple images. Images were taken of the FITC concentrations in a well plate with varying settings. Exposure time and brightness level were changed to see which sets of controls can best represent the trend between FITC concentration and pixel intensity. The final experiment cycled through nine different exposure times, ranging from 1 ms to 10000ms. For each exposure time, five images of the well plate were captured with a brightness level of 1. For mathematical analysis, a python script was used to save the green-channel intensities for each of the filled wells, along with two empty wells for comparison. At each well, the script took the mean intensity and standard deviation of a circle of pixels within a radius of 20 pixels, when given the corresponding center pixel location. These values were saved in an excel spreadsheet and then inputted into Matlab to find the mean and standard deviation of each concentration of FITC across five sets of images. The values were then plotted as intensity over FITC concentration in M.

### 4.3.3 Laser Speckle Contrast Imaging

In order to verify the analysis and LSCI computation of speckle images, a python script was used to create simulated speckle images. A simulated speckle pattern was generated using Python's numpy library to make an NxN array of randomized, complex intensity values. Typical RGB images have intensities ranging from 0 to 255 arbitrary units. Given the simulated speckle array, a python processing script was used to calculate the phase and amplitude information of the image. Additionally, the script applied a circular, low pass filter in the fourier domain to the speckle pattern to enhance the quality of the image. An autocorrelation calculation was then applied to both the original and filtered speckle patterns using the fourier method described in Section 1.3 in order to quantify the temporal or spatial variations. Next, the speckle size was determined by finding the location in which the autocorrelation reaches the maximum of its peak on either side. The simulation was then run through a LSCI loop to calculate the K values as described in Section 1.3.

To take real-world speckle images for analysis, the modified box lid described in Section 2.3.2 was used. The laser module in Fig. #a was affixed to the lid of the OpenIVIS box using a 3/8" 1/4-20 bolt along with the Arducam camera, as shown in Fig. #b. In order to mimic movement that might be seen in a biological system, such as a vein with blood flow, a clear plastic tube was run through the center of the box. Diffuse liquid was pushed through the tube at varying flow rates by a kdScientific Syringe Pump and a 20 mL syringe with a Luer Lock. The area of the tube under study was placed directly under the Raspberry Pi camera and

illuminated with the laser module. Images were taken using the corresponding python script and saved as .bmp files, and Bayer images as .npy arrays. Additionally, the green channel of the Bayer image were saved using a mask, as discussed in Section 2.3.2. The resulting images were processed using an additional python program to calculate the speckle size, intensity histogram and LSCI pattern.

### 4.3.4 Additional Capabilities

In addition to fluorescence imaging and LSCI, the OpenIVIS system was verified for time lapse imaging. As discussed in the paper by Branning et. al [Branning], the IVIS system can be used to image growth or changes in an organism or process over time. To verify this capability, 11 Tobacco Hornworms in individual, clear tubes were placed into the main section of the imaging box. The box was kept in a room set to 73oC to maintain healthy temperatures. The Neopixel LEDs were set to their full brightness in order to promote proper lighting conditions for growth. A python script was run through the Raspberry Pi that captures time lapsed images over a 10 day period. Images were set to be taken every five hours during that period. The images were then saved in a folder to be analyzed for hornworm growth.

The OpenIVIS system's versatile nature also provides opportunities for further scientific experiments. One possibility for the system's applications is the imaging of a degrading plant's Anthocyanin response. When plants decompose, they produce a fluorescent pigment called Anthocyanin. The Anthocyanin response can be visualized by taking the logarithm of the pixel intensities when excited by a red wavelength of light, over the intensity excited by a green wavelength of light, as shown in Eq. 5. In the OpenIVIS box, a lettuce leaf was placed in the main compartment. Using the Neopixel LED array, images were taken with red and green excitations of 653 nm and 520 nm respectively. Additionally, a long pass optical filter with a cutoff wavelength of 665 nm was placed below the camera in the system's filter holder to remove wavelengths below the red value. Images were taken three days apart for decomposition analysis. The images were then processed in python by applying Eq. 5 for each pixel location in the image.

$$A_R = log(\frac{I_{635nm} - I_{520nm}}{I_{635nm} + I_{520nm}}) \quad \text{(Eq.5)} \tag{4.1}$$

# 5 Results

## 5.1 System Verification and Setup

### 5.1.1 Changing Exposure Time

In Fig. 5.1, there is a positive, non-linear relationship between exposure time and pixel intensity. Initially there is a large increase in pixel intensity as exposure time increases and this is most likely because as exposure time increases, more photons will be detected by the pixels in the camera. As exposure time continues to increase, the increase in pixel intensity starts to decrease. This is most likely because the pixels can only hold a certain amount of photons and when a pixel has stored its maximum amount of photons, it has reached its saturation point. As more photons are being detected by the pixels from the increasing exposure times, the pixels are getting closer to their saturation point [5]. This plot also has a small error across the different trials, suggesting that the exposure times are precise.

#### Read and Plot Exposure Values

```
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
from scipy.optimize import curve_fit

data1 = pd.read_excel("Data/exposure/lastExposure1.xlsx", sheet_name="Sheet2")
data2 = pd.read_excel("Data/exposure/lastExposure2.xlsx", sheet_name="Sheet2")
data3 = pd.read_excel("Data/exposure/lastExposure3.xlsx", sheet_name="Sheet2")
data4 = pd.read_excel("Data/exposure/lastExposure4.xlsx", sheet_name="Sheet2")
data5 = pd.read_excel("Data/exposure/lastExposure5.xlsx", sheet_name="Sheet2")

# For Exposure Times
data1_subsets = [data1.iloc[i:i+8].values.flatten() for i in range(0, 45, 9)]
data2_subsets = [data2.iloc[i:i+8].values.flatten() for i in range(0, 45, 9)]
data3_subsets = [data3.iloc[i:i+8].values.flatten() for i in range(0, 45, 9)]
data4_subsets = [data4.iloc[i:i+8].values.flatten() for i in range(0, 45, 9)]
```

```
data5_subsets = [data5.iloc[i:i+8].values.flatten() for i in range(0, 45, 9)]

# Calculate means and std for each subset
m_data1 = np.array([np.mean(subset) for subset in data1_subsets])
m_data2 = np.array([np.mean(subset) for subset in data2_subsets])
m_data3 = np.array([np.mean(subset) for subset in data3_subsets])
m_data4 = np.array([np.mean(subset) for subset in data4_subsets])
m_data5 = np.array([np.mean(subset) for subset in data5_subsets])

s_data1 = np.array([np.std(subset) for subset in data1_subsets])
s_data2 = np.array([np.std(subset) for subset in data2_subsets])
s_data3 = np.array([np.std(subset) for subset in data3_subsets])
s_data4 = np.array([np.std(subset) for subset in data4_subsets])
s_data5 = np.array([np.std(subset) for subset in data5_subsets])
#
# Combine means and std of each numbered data
means = np.mean([m_data1, m_data2, m_data3, m_data4, m_data5], axis=0)

stds = np.std([s_data1, s_data2, s_data3, s_data4, s_data5], axis=0)

exp = [0.001, 0.005, 0.01, 0.05, 0.1]

plt.figure()
plt.errorbar(exp, means, yerr=stds, fmt='o', color='red', linewidth=2, markersize=6, label='
plt.title('Brightness Level = 0.7')
plt.suptitle('Pixel Intensity vs. Exposure Time')
plt.ylim([0, 1100])
plt.xlabel('Exposure Time [s]')
plt.ylabel('Pixel Intensity [a.u.]')
# Polynomial Fit
coeffs = np.polyfit(exp, means, 4)
y_fit = np.polyval(coeffs, exp)
plt.plot(exp, y_fit, 'r', linewidth=2)

plt.savefig('images/exposure.png')
plt.close()
```

### 5.1.2 Changing Brightness Level

In both of the plots shown in Fig. 5.2, there is a positive trend between brightness level and pixel intensity. When the exposure time was set to 0.01 seconds (Fig. # a), the pixel intensity

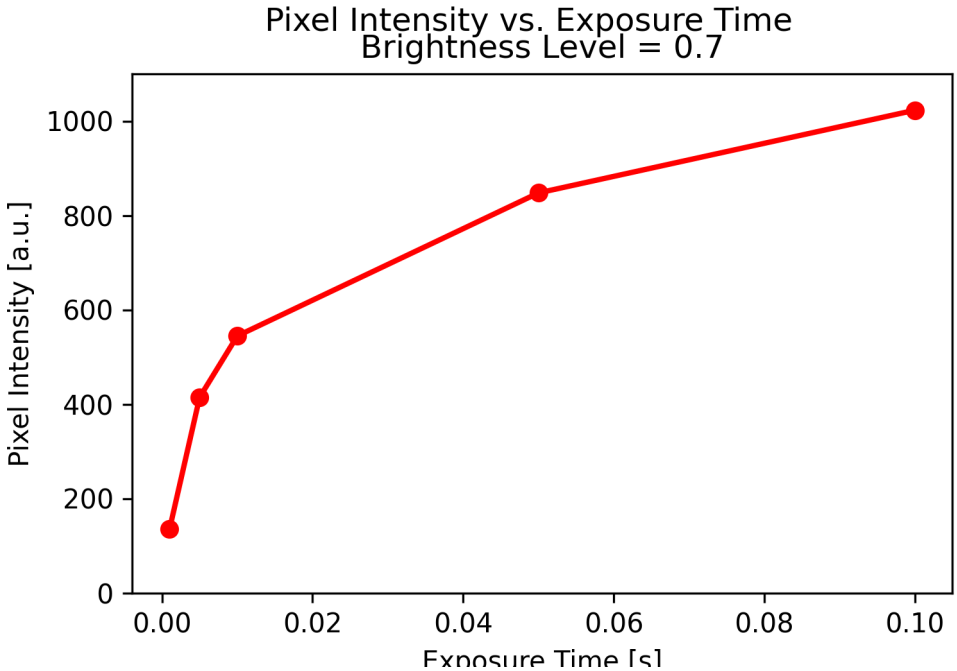


Figure 5.1: The pixel intensity versus exposure time.

increases linearly and then flattens out around 1023 arbitrary units. This number matches with the intensity when exposure time is 0.01 seconds in Fig. 5.1 above, suggesting that the exposure time has a significant impact on the resulting plot of brightness level vs pixel intensity. When the exposure time is increased to 0.1 seconds in Fig. # b, the linear portion of the graph seems to shift to higher brightness levels and starts to flatten out around 1000 arbitrary units. This intensity value also agrees with the intensity value of an exposure time of 0.1 seconds in Fig. # above. The trend that this data supports is that the camera will most accurately capture LED brightness when the brightness level is low and the exposure time is quick and when the brightness level is high and the exposure time is slow. There is also a very small error across the different trials, suggesting that the camera is capturing the LED brightness precisely.

### Read and Plot Brightness Values

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Read the data from Sheet 2
```

```
data1 = pd.read_excel("Data/brightness/lastBrightness1.xlsx", sheet_name="Sheet2")
data2 = pd.read_excel("Data/brightness/lastBrightness2.xlsx", sheet_name="Sheet2")
data3 = pd.read_excel("Data/brightness/lastBrightness3.xlsx", sheet_name="Sheet2")
data4 = pd.read_excel("Data/brightness/lastBrightness4.xlsx", sheet_name="Sheet2")
data5 = pd.read_excel("Data/brightness/lastBrightness5.xlsx", sheet_name="Sheet2")

# For Brightness Levels
data1_subsets = [data1.iloc[i:i+9].values.flatten() for i in range(0, 99, 9)]
data2_subsets = [data2.iloc[i:i+9].values.flatten() for i in range(0, 99, 9)]
data3_subsets = [data3.iloc[i:i+9].values.flatten() for i in range(0, 99, 9)]
data4_subsets = [data4.iloc[i:i+9].values.flatten() for i in range(0, 99, 9)]
data5_subsets = [data5.iloc[i:i+9].values.flatten() for i in range(0, 99, 9)]

# Calculate means and std for each subset
m_data1 = np.array([np.mean(subset) for subset in data1_subsets])
m_data2 = np.array([np.mean(subset) for subset in data2_subsets])
m_data3 = np.array([np.mean(subset) for subset in data3_subsets])
m_data4 = np.array([np.mean(subset) for subset in data4_subsets])
m_data5 = np.array([np.mean(subset) for subset in data5_subsets])

s_data1 = np.array([np.std(subset) for subset in data1_subsets])
s_data2 = np.array([np.std(subset) for subset in data2_subsets])
s_data3 = np.array([np.std(subset) for subset in data3_subsets])
s_data4 = np.array([np.std(subset) for subset in data4_subsets])
s_data5 = np.array([np.std(subset) for subset in data5_subsets])

# Combine means and std of each numbered data
means = np.mean([m_data1, m_data2, m_data3, m_data4, m_data5], axis=0)

stds = np.std([s_data1, s_data2, s_data3, s_data4, s_data5], axis=0)

bri = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]

# Plotting
plt.figure()
plt.errorbar(bri, means, yerr=stds, fmt='o', color='blue', linewidth=2, markersize=6, label=
plt.title('Exposure Time = 0.1 s')
plt.suptitle('Pixel Intensity vs. Brightness Level')
plt.ylim([0, 1100])
plt.xlabel('Brightness Level [a.u.]')
plt.ylabel('Pixel Intensity [a.u.]')
# Polynomial Fit
```

```
coeffs = np.polyfit(bri, means, 4)
y_fit = np.polyval(coeffs, bri)
plt.plot(bri, y_fit, 'b', linewidth=2)

plt.savefig('images/brightness.png')
plt.close()
```

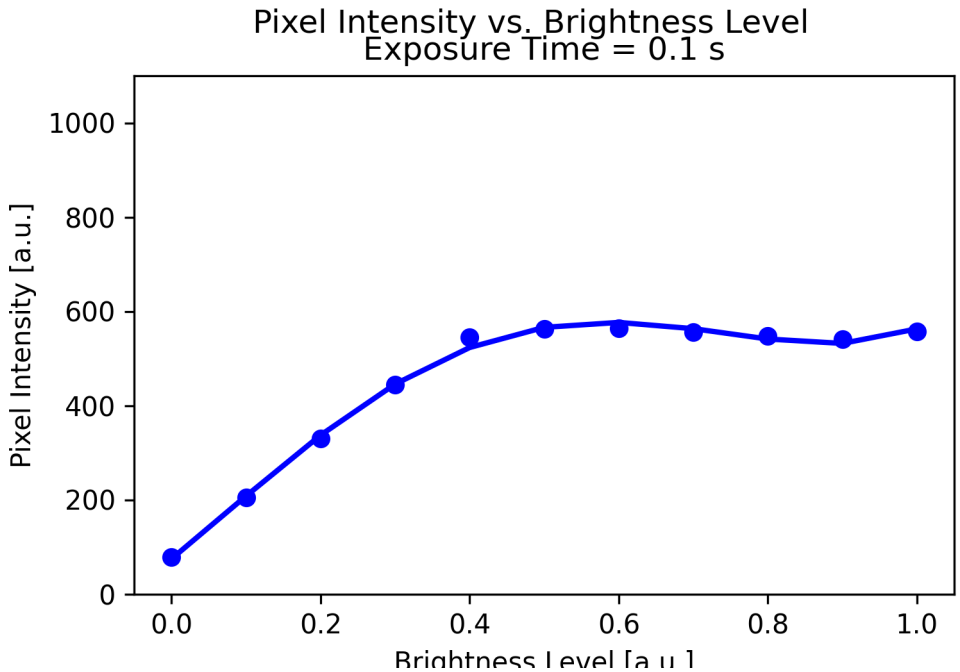


Figure 5.2: The pixel intensity versus brightness level.

### 5.1.3 Measuring From a Photodiode

Both Fig. 5.3 and 5.4 depict a positive, non-linear relationship between brightness and wattage from the photodiode. Fig. 5.3 a shows increasing brightness level from the NeoPixel controls while Figure 5.4 shows increasing tuple value, or color code value. The plots are extremely similar in both shape and in error. The error across the different trials are a lot larger than in Figuress 5.1 and 5.2. This is because the photodiode is measuring the actual LED output rather than what the camera is capturing the brightness to be. The LEDs being used are off the shelf and relatively cheap, so it was expected that they would be less precise than more expensive LED options.

### Read and Plot Photodiode Values

```
import numpy as np
import matplotlib.pyplot as plt

# Data
level = [0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1]
meanLevel = [0.566, 194.9, 373.5, 552, 687, 820, 964.3333333, 1063.333333, 1152, 1222, 1287.3
stdLevel = [0.305, 20.22374842, 37.05738793, 56.15158057, 57.10516614, 67.44627492, 97.42860

tuple_vals = [0, 25, 51, 76, 102, 127, 153, 178, 204, 229, 255]
meanTuple = [0.566, 194.7, 384, 552, 710, 843, 961.3333333, 1061, 1148.333333, 1221, 1287.333
stdTuple = [0.305, 19.43167517, 38.19685851, 53.3291665, 67.29041537, 76.92203845, 83.912653

# Plot for tuple values
plt.figure()
plt.errorbar(tuple_vals, meanTuple, yerr=stdTuple, fmt='ob', linewidth=2, markersize=6, marke

# Polynomial fit for tuple values
a1 = np.polyfit(tuple_vals, meanTuple, 2)
y1 = np.polyval(a1, tuple_vals)
plt.plot(tuple_vals, y1, 'b', linewidth=2)
pStr1 = 'y = {:.2f}x^2 + {:.2f}x + {:.2f}'.format(a1[0], a1[1], a1[2])
# Calculate the residuals
residuals1 = meanTuple - y1

# Calculate the total sum of squares (SStot)
SStot1 = np.sum((meanTuple - np.mean(meanTuple))**2)

# Calculate the sum of squares of residuals (SSres)
SSres1 = np.sum(residuals1**2)

# Calculate R-squared
R21 = 1 - (SSres1 / SStot1)

# Display the results
pStr1R = f'R² = {R21:.4f}'

plt.title('LED Output vs. Tuple Value for Brightness Level = 1')
plt.xlabel('Tuple Value [a.u.]')
plt.ylabel('LED Output [nW]')
plt.xlim([0, 260])
```

```
plt.ylim([0, 1400])
xPos = max(tuple_vals) - 150
yPos = min(meanTuple) + 200
plt.text(xPos, yPos, pStr1, fontsize=12, color='b')
plt.text(xPos, yPos-100, pStr1, fontsize=12, color='b')

plt.legend()
plt.savefig('images/photodiodeTuple.png')
plt.close()

# Plot for brightness levels
plt.figure()
plt.errorbar(level, meanLevel, yerr=stdLevel, fmt='+r', linewidth=2, markersize=6, markerfac

# Polynomial fit for brightness levels
a2 = np.polyfit(level, meanLevel, 2)
y2 = np.polyval(a2, level)
plt.plot(level, y2, 'r', linewidth=2)
pStr2 = 'y = {:.2f}x^2 + {:.2f}x + {:.2f}'.format(a2[0], a2[1], a2[2])
# Calculate the residuals
residuals2 = meanLevel - y2
# Calculate the total sum of squares (SStot)
SStot2 = np.sum((meanLevel - np.mean(meanLevel))**2)
# Calculate the sum of squares of residuals (SSres)
SSres2 = np.sum(residuals2**2)
# Calculate R-squared
R22 = 1 - (SSres2 / SStot2)
# Display the results
pStr2R = f'R² = {R22:.4f}'

plt.title('LED Output vs. Brightness Level for Brightness Tuple = (0,0,0,255)')
plt.xlabel('Brightness Level [a.u.]')
plt.ylabel('LED Output [nW]')
plt.xlim([0, 1.1])
plt.ylim([0, 1400])
xPos = max(level) - 0.6
yPos = min(meanLevel) + 200
plt.text(xPos, yPos, pStr2, fontsize=12, color='r')
plt.text(xPos, yPos-100, pStr2, fontsize=12, color='r')
plt.legend()

plt.savefig('images/photodiodeLevel.png')
```

```
plt.close()
```

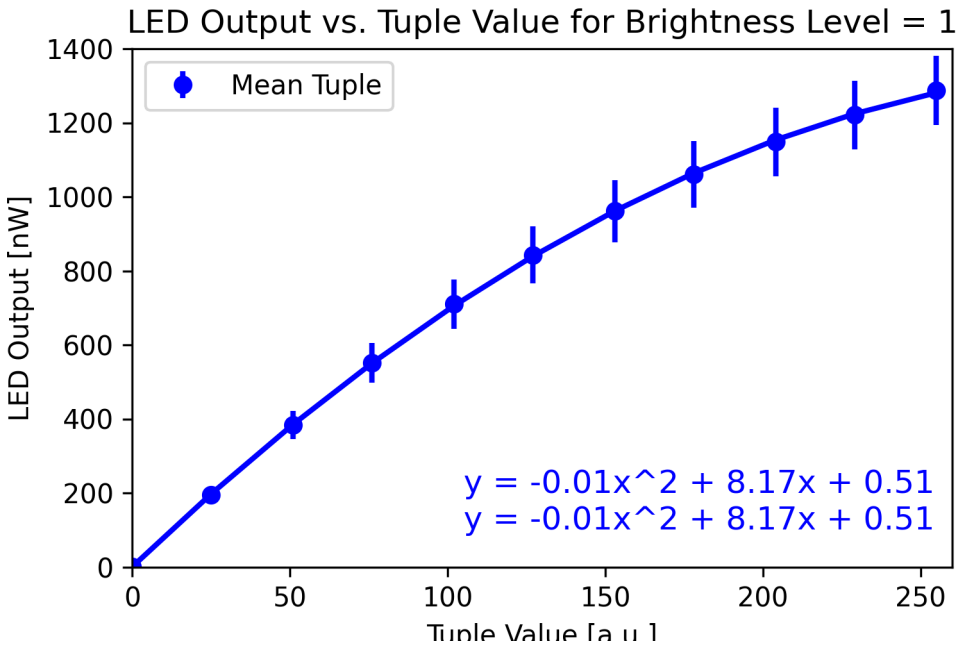


Figure 5.3: The photodiode pixel intensity versus color code for a brightness level of 1.

## 5.2 Flourescence Imaging

Fig. # shows the normalized pixel intensities of all ten concentrations of FITC for eight different exposure times. As exposure time and concentration increases, the pixel intensity also increases. The 100 milliseconds, or 0.1 seconds, exposure time best represents the expected relationship between concentration and pixel intensity (green curve). As seen in Fig. #, the lower concentrations of FITC are a lot less fluorescent than the higher concentrations. The green curve shows a large increase in pixel intensity in the higher concentrations, with it flattening out at the maximum value of 1 in the normalized plot. This maximum value of 1 corresponds to 1023 arbitrary units. The slower exposure times (purple and blue curves) also show the trend to an extent. These larger exposure times mean that the images captured by the camera are super bright, so most of the concentrations have large intensity values. The faster exposure times have the opposite effect. The images captured by the camera are super dark, and the pixel intensity values are low. The intensity values for each concentration represent how fluorescent each concentration is. Based on this data, the higher concentrations of FITC are more fluorescent than the lower concentrations.

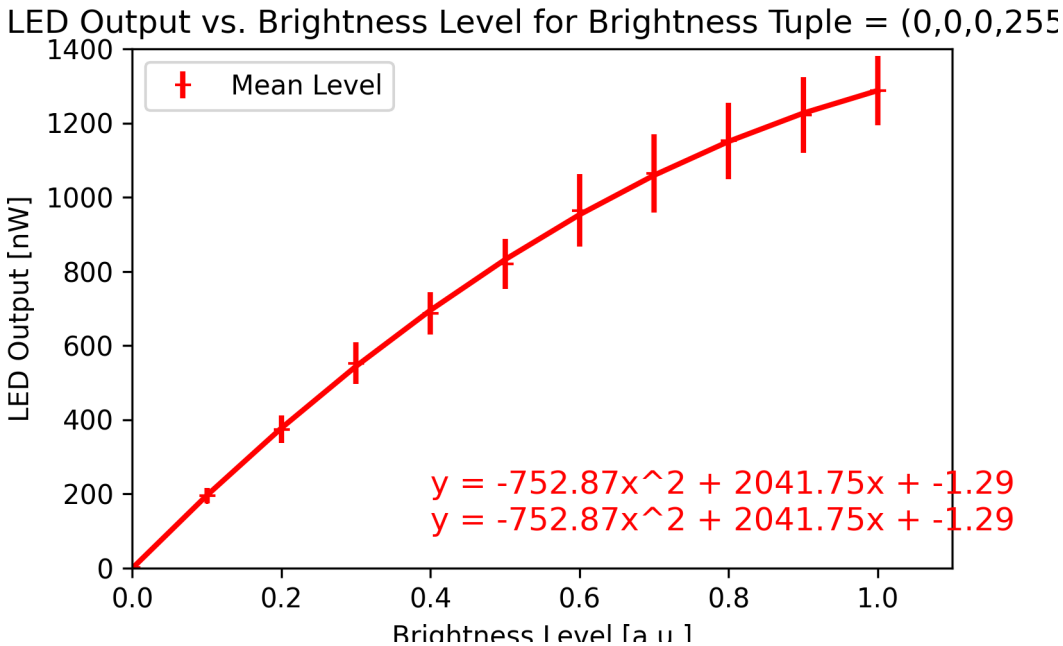


Figure 5.4: The photodiode pixel intensity versus brightness level for a color code of (0,0,0,255)

**Read FITC Data and Extract Values**

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

# Read the data from the Excel files
data1 = pd.read_excel("Data/FITC/pixel_circle1.xlsx")
data2 = pd.read_excel("Data/FITC/pixel_circle2.xlsx")
data3 = pd.read_excel("Data/FITC/pixel_circle3.xlsx")
data4 = pd.read_excel("Data/FITC/pixel_circle4.xlsx")
data5 = pd.read_excel("Data/FITC/pixel_circle5.xlsx")

data1m = data1.iloc[:, 2].values
data2m = data2.iloc[:, 2].values
data3m = data3.iloc[:, 2].values
data4m = data4.iloc[:, 2].values
data5m = data5.iloc[:, 2].values

data1s = data1.iloc[:, 3].values
```

```
data2s = data2.iloc[:, 3].values
data3s = data3.iloc[:, 3].values
data4s = data4.iloc[:, 3].values
data5s = data5.iloc[:, 3].values

# Extract the subsets For FITC
m_con_00001 = np.array([data1m[:10], data2m[:10], data3m[:10], data4m[:10], data5m[:10]])
m_con_00005 = np.array([data1m[10:21], data2m[10:21], data3m[10:21], data4m[10:21], data5m[10
m_con_00010 = np.array([data1m[21:32], data2m[21:32], data3m[21:32], data4m[21:32], data5m[21
m_con_00050 = np.array([data1m[32:43], data2m[32:43], data3m[32:43], data4m[32:43], data5m[32
m_con_00100 = np.array([data1m[43:54], data2m[43:54], data3m[43:54], data4m[43:54], data5m[43
m_con_00500 = np.array([data1m[54:65], data2m[54:65], data3m[54:65], data4m[54:65], data5m[54
m_con_01000 = np.array([data1m[65:76], data2m[65:76], data3m[65:76], data4m[65:76], data5m[65
m_con_10000 = np.array([data1m[76:87], data2m[76:87], data3m[76:87], data4m[76:87], data5m[76
m_con_05000 = np.array([data1m[87:98], data2m[87:98], data3m[87:98], data4m[87:98], data5m[87

m_con_00001 = m_con_00001.astype(float)
m_con_00005 = m_con_00005.astype(float)
m_con_00010 = m_con_00010.astype(float)
m_con_00050 = m_con_00050.astype(float)
m_con_00100 = m_con_00100.astype(float)
m_con_00500 = m_con_00500.astype(float)
m_con_01000 = m_con_01000.astype(float)
m_con_10000 = m_con_10000.astype(float)
m_con_05000 = m_con_05000.astype(float)

# Calculate standard deviations
std_con_00001 = np.std(m_con_00001, axis=0)
std_con_00005 = np.std(m_con_00005, axis=0)
std_con_00010 = np.std(m_con_00010, axis=0)
std_con_00050 = np.std(m_con_00050, axis=0)
std_con_00100 = np.std(m_con_00100, axis=0)
std_con_00500 = np.std(m_con_00500, axis=0)
std_con_01000 = np.std(m_con_01000, axis=0)
std_con_05000 = np.std(m_con_05000, axis=0)
std_con_10000 = np.std(m_con_10000, axis=0)

# Calculate means
m_con_00001 = np.mean(m_con_00001, axis=0)
m_con_00005 = np.mean(m_con_00005, axis=0)
m_con_00010 = np.mean(m_con_00010, axis=0)
m_con_00050 = np.mean(m_con_00050, axis=0)
```

```
m_con_00100 = np.mean(m_con_00100, axis=0)
m_con_00500 = np.mean(m_con_00500, axis=0)
m_con_01000 = np.mean(m_con_01000, axis=0)
m_con_05000 = np.mean(m_con_05000, axis=0)
m_con_10000 = np.mean(m_con_10000, axis=0)

# Normalize data
std_con_00001 /= 1023
std_con_00005 /= 1023
std_con_00010 /= 1023
std_con_00050 /= 1023
std_con_00100 /= 1023
std_con_00500 /= 1023
std_con_01000 /= 1023

m_con_00001 /= 1023
m_con_00005 /= 1023
m_con_00010 /= 1023
m_con_00050 /= 1023
m_con_00100 /= 1023
m_con_00500 /= 1023
m_con_01000 /= 1023
```

### Plot FITC Data

```
# Plotting
concentration = [10**-3, 10**-4, 10**-5, 10**-6, 10**-7, 10**-8, 10**-9, 10**-10, 10**-11, 1
conc = [-3, -4, -5, -6, -7, -8, -9, -10, -11, -12]

plt.plot(conc, m_con_00001[:10], '-o', color='blue', linewidth=1, markersize=6, markerfaceco
plt.plot(conc, m_con_00005[:10], '-o', color='magenta', linewidth=1, markersize=6, markerfac
plt.plot(conc, m_con_00010[:10], '-o', color='#FFA500', linewidth=1, markersize=6, markerfac
plt.plot(conc, m_con_00050[:10], '-o', color='red', linewidth=1, markersize=6, markerfacecol
plt.plot(conc, m_con_00100[:10], '-o', color='green', linewidth=1, markersize=6, markerfacec
plt.plot(conc, m_con_00500[:10], '-o', color='cyan', linewidth=1, markersize=6, markerfaceco
plt.plot(conc, m_con_01000[:10], '-o', color='#7E2F8E', linewidth=1, markersize=6, markerfac

plt.errorbar(conc, m_con_00001[:10], yerr=std_con_00001[:10], fmt='o', color='blue', linewid
plt.errorbar(conc, m_con_00005[:10], yerr=std_con_00005[:10], fmt='o', color='magenta', line
plt.errorbar(conc, m_con_00010[:10], yerr=std_con_00010[:10], fmt='o', color='#FFA500', line
```

```
plt.errorbar(conc, m_con_00050[:10], yerr=std_con_00050[:10], fmt='o', color='red', linewidth
plt.errorbar(conc, m_con_00100[:10], yerr=std_con_00100[:10], fmt='o', color='green', linewi
plt.errorbar(conc, m_con_00500[:10], yerr=std_con_00500[:10], fmt='o', color='cyan', linewidt
plt.errorbar(conc, m_con_01000[:10], yerr=std_con_01000[:10], fmt='o', color='#7E2F8E', linew

plt.title('Pixel Intensity vs. Concentration')
plt.xlabel('Log[FITC Concentration M]')
plt.ylabel('Pixel Intensity')
plt.legend(loc='upper right')
plt.savefig('images/FITC.png')
plt.close()
```

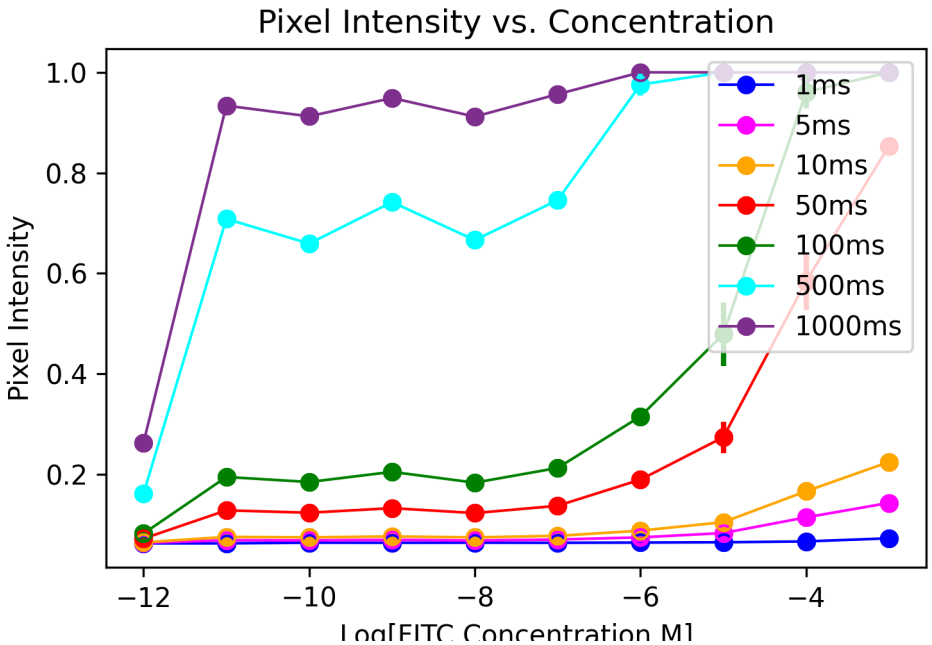


Figure 5.5: The pixel intensity versus FITC concentration for varying exposure times.

## 5.3 Laser Speckle Contrast Imaging

### 5.3.1 Simulation

Using the process described in Section 3.3.3, analysis on simulated speckle images and the effect of filtering was run. Fig. # depicts the simulated results for a filter with cutoff frequency of

N4, where N denotes the size of the speckle image in pixels, while Figures # and # show the results for cutoff frequency of N8 and N16, respectively.

### Create a Simulated Speckle Image

```
# Import the necessary Python packages
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import os
from PIL import Image

def ft2(im):
    """
    Takes the fourier transform
    """
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(im)))

# Function for Inverse Fourier Transform
def ift2(im):
    """
    Takes the inverse fourier transform
    """
    return np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(im)))

# Generate a speckle pattern
N = 255  # Size of the speckle pattern
a = np.random.randn(N, N) + 1j * np.random.randn(N, N)
```

### Create a filter and Apply it to the Speckle Image

```
# Compute amplitude, phase and intensity of the original speckle pattern
amplitude = np.abs(a)
phase = np.angle(a)
intensity = amplitude **2

# Fourier transform of the speckle field
A = ft2(a)
# Create a circular low-pass filter
```

```
x = np.linspace(-N/2, N/2, N)
y = np.linspace(-N/2, N/2, N)
X, Y = np.meshgrid(x, y)
radius = np.sqrt(X**2 + Y**2)
cutoff = N/4  # Adjust the cutoff frequency to control the speckle size
filter = radius < cutoff

# Apply the low-pass filter
A_filtered = A * filter

# Inverse Fourier transform back to the spatial domain
a_filtered = ift2(A_filtered)


# Compute the amplitude, phase and intensity of the filtered speckle pattern
amplitude_filtered = np.abs(a_filtered)
phase_filtered = np.angle(a_filtered)
intensity_filtered = amplitude_filtered **2
```

**Compute the Autocorrelation and Speckle Size**

```
# Compute autocorrelation of the original and filtered speckle patterns
autocorrelation_original = ift2(ft2(intensity) * np.conj(ft2(intensity)))
autocorrelation_filtered = ift2(ft2(intensity_filtered) * np.conj(ft2(intensity_filtered)))

autocorrelation_original = np.abs(autocorrelation_original)  # Take the magnitude
autocorrelation_filtered = np.abs(autocorrelation_filtered)  # Take the magnitude

# Normalize the autocorrelations for better visualization
autocorrelation_original /= autocorrelation_original.max()
autocorrelation_filtered /= autocorrelation_filtered.max()

# The code below is for finding the speckle sizes of the images

# Get the slice of the autocorrelation in line with the peak
central_slice_original = autocorrelation_original[N//2, :]
central_slice_filtered = autocorrelation_filtered[N//2, :]

# Find the halfway point between the maximum and the steady state
half_max_original = ((central_slice_original.max() - central_slice_original.min()) / 2 ) + c
```

```
half_max_filtered = ((central_slice_filtered.max() - central_slice_filtered.min()) / 2 ) + c

# Find all of the pixels above the halfway point
indices_original = np.where(central_slice_original >= half_max_original)[0]
indices_filtered = np.where(central_slice_filtered >= half_max_filtered)[0]

# Find the width of the peak at the halfway point
if indices_original.size == 1:
    speckle_size_original = 1
else:
    speckle_size_original = indices_original[-1] - indices_original[0]

if indices_filtered.size == 1:
    speckle_size_filtered = 1
else:
    speckle_size_filtered = indices_filtered[-1] - indices_filtered[0]

# Print the speckle sizes
print(f"Original speckle size (width at half maximum): {speckle_size_original} pixels")
print(f"Filtered speckle size (width at half maximum): {speckle_size_filtered} pixels")


# Rayleigh Distribution for the amplitude histograms
flat = np.abs(amplitude.flatten())
flat_fil = np.abs(amplitude_filtered.flatten())

sigma = np.sqrt(np.mean(flat**2) / 2)
sigma_fil = np.sqrt(np.mean(flat_fil**2) / 2)

x = np.linspace(0, np.max(flat), 255)
x_fil = np.linspace(0, np.max(flat_fil), 255)

ray = (x / sigma**2) * np.exp(-x**2 / (2 * sigma**2))
ray_fil = (x_fil / sigma_fil**2) * np.exp(-x_fil**2 / (2 * sigma_fil**2))


# Zoom into the autocorrelations
zoom = 100
center = np.array(autocorrelation_original.shape) // 2
center_fil = np.array(autocorrelation_filtered.shape) // 2

zoomRegion = (slice(center[0]-zoom, center[0]+zoom), slice(center[1]-zoom, center[1]+zoom))
```

```
zoomRegion_fil = (slice(center_fil[0]-zoom, center_fil[0]+zoom), slice(center_fil[1]-zoom, ce

autoZoom = autocorrelation_original[zoomRegion]
autoZoom_fil = autocorrelation_filtered[zoomRegion_fil]
```

```
Original speckle size (width at half maximum): 247 pixels
Filtered speckle size (width at half maximum): 1 pixels
```

### Create plots

```
def add_colorbar(him, ax, cbar_title=""):
    """
    This function adds a nicely-formatted colorbar
    """
    divider = make_axes_locatable(ax)
    cax = divider.append_axes("right", size="5%", pad=0.05)
    cbar = plt.colorbar(him, cax=cax)
    cbar.set_label(cbar_title, rotation=270, labelpad=15)

# Fiure fo comparing the autocorrelations of the original and filtered speckle patterns
fig, a = plt.subplots(ncols=2, nrows=3, figsize=(10,20))

# Plot the original amplitude
im_amp = a[0, 0].imshow(amplitude, cmap='gray')
a[0, 0].set_title('Original Speckle Field Amplitude')
add_colorbar(im_amp, a[0, 0], "Amplitude [a.u.]")

# Plot the filtered amplitude
im_amp_filtered = a[0, 1].imshow(amplitude_filtered, cmap='gray')
a[0, 1].set_title('Filtered Speckle Field Amplitude')
add_colorbar(im_amp_filtered, a[0, 1], "Amplitude [a.u.]")

# Plot the autocorrelation of the original speckle pattern
im_autoZoom = a[1, 0].imshow(autoZoom, cmap='gray')
a[1, 0].set_title('Autocorrelation of Original Speckle Pattern')
add_colorbar(im_autoZoom, a[1, 0], "Intensity [a.u.]")

# Plot the autocorrelation of the filtered speckle pattern
im_autoZoom_fil = a[1, 1].imshow(autoZoom_fil, cmap='gray')
a[1, 1].set_title('Autocorrelation of Filtered Speckle Pattern')
```

```
add_colorbar(im_autoZoom_fil, a[1, 1], "Intensity [a.u.]")

# Plot the central slice of the original autocorrelation (zoomed)
a[2,0].plot(central_slice_original, color='black')
a[2,0].axhline(y=half_max_original, color='red', linestyle='--')
a[2,0].set_title('Central Slice of Original Autocorrelation')
a[2,0].set_xlabel('Pixel')
a[2,0].set_ylabel('Intensity [a.u.]')
a[2,0].text(0.5, 0.9, f'Speckle size: {speckle_size_original} pixels', color='red', transfor

# Plot the central slice of the filtered autocorrelation (zoomed)
a[2,1].plot(central_slice_filtered, color='black')
a[2,1].axhline(y=half_max_filtered, color='red', linestyle='--')
a[2,1].set_title('Central Slice of Filtered Autocorrelation')
a[2,1].set_xlabel('Pixel')
a[2,1].set_ylabel('Intensity [a.u.]')
a[2,1].text(0.5, 0.9, f'Speckle size: {speckle_size_filtered} pixels', color='red', transfor

# Adjust layout
plt.tight_layout()
# Save the plot
plt.savefig('images/LSCISimulation.png')
plt.close(fig)
```

As the cutoff frequency decreases, the size of the speckle in pixels increases. The original speckle pattern has a speckle size of 1. A cutoff frequency of N4 results in a filtered speckle size of 2 pixels, N8 results in a filtered speckle size of 4 pixels, and N16 results in a filtered speckle size of 8 pixels, as shown in Figures #x, #x, and #x. This implies that the filter size directly affects the resulting speckle size and a more narrow filter will produce a lower resolution image, but may decrease the noise in the speckle pattern. Additionally, as the filter's cutoff frequency decreases, the amplitude histogram of the filtered patterns intensity decreases. Fig. #x shows a normalized amplitude peak at approximately 1 for a cutoff frequency of N4, while Fig. #x shows an amplitude peak at 0.1 for a cutoff frequency of N16. The simulated results also show an increase in autocorrelation with a decrease in cutoff frequency, as shown in Figures #x, #x, and #x. The change in autocorrelation is indicative of the change in speckle size across the simulations. In real-world applications, an increase in autocorrelation across images may also imply differences in the subject's roughness or uniformity.

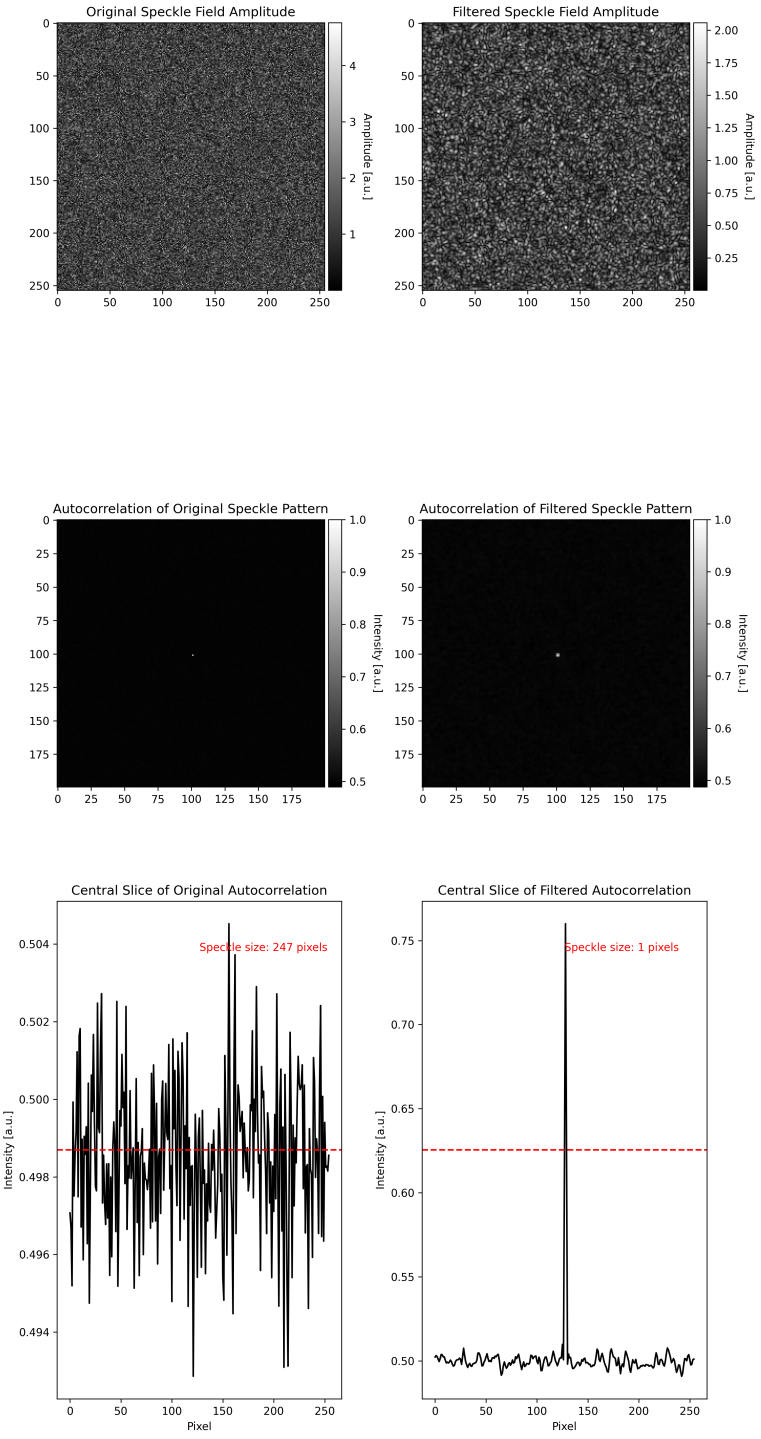


Figure 5.6: The simulated and filtered speckle images.

### 5.3.2 Physical Tests

The LSCI analysis script was run on images with a cream and water mixture running through the tube at flow rates of 0 ml/hr, 50 ml/hr, 100 ml/hr, 150 ml/hr, and 200 ml/hr. The resulting speckle images, LSCI reproductions and LSCI K values are shown in Fig. 5.7 and Table 5.1. While there is a general decreasing trend in the K values as flow rate decreases, there are discrepancies from value to value.

#### Finding the moving LSCI K value

```
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.axes_grid1 import make_axes_locatable
import cv2
from scipy.ndimage import gaussian_filter
import os
import pandas as pd

# Load BMP image and convert to grayscale numpy array
def load_and_normalize_image(image_path):
    img = Image.open(image_path).convert('L')
    img_array = np.array(img)
    # Normalize the image to range [0, 1]
    img_normalized = img_array / 255.0
    return img_normalized

# Function for Fourier Transform
def ft2(im):
    return np.fft.fftshift(np.fft.fft2(np.fft.ifftshift(im)))

# Function for Inverse Fourier Transform
def ift2(im):
    return np.fft.ifftshift(np.fft.ifft2(np.fft.fftshift(im)))

# Calculate speckle contrast
def calcSpeckleContrast(image_path):
    x1 = 2150; x2 = 2600
    y1 = 1200; y2 = 1600

    image = plt.imread(image_path)
```

```
    intensity = np.asarray(image[y1:y2, x1:x2])
    photo = gaussian_filter(intensity, 2)

    window = photo[:][:]
    w_dev = np.std(window)
    w_meanInt = np.mean(window)
    w_speckle_contrast = w_dev / w_meanInt

    # the selection of SQUARE x SQUARE pixels where speckle will be calculated
    SQUARE = 7

    # convert photo into an np array
    pArr = np.asarray(photo)

    # the speckle contrast array that will be returned
    contrast_array = [ [0]*(len(pArr[0])-SQUARE) for i in range((len(pArr))-SQUARE) ]
    conMin = np.inf;

    # The Loop Does the Following:
    # goes through 7x7 selections of the image
    # computes the speckle contrast for this region
    # using this comparison, create a new matrix with values whose magnitude
        # decreases depending on how "blurry" the selection is
    for r in range(len(pArr) - SQUARE):
        for c in range(len(pArr[0]) - SQUARE):
            x1 = r; x2 = r+SQUARE-1
            y1 = c; y2 = c+SQUARE-1
            selection = pArr[x1:x2, y1:y2]

            s_dev = np.std(selection)
            s_meanInt = np.mean(selection)

            # to cut out values outside of the speckle caught on camera
            #if (s_meanInt <= 0.1 and s_dev <= 0.1):
              # contrast_array[r][c] = np.NaN
            #else:
            s_speckle_contrast = s_dev / s_meanInt

            if s_speckle_contrast < conMin and s_speckle_contrast != 0:
                conMin = s_speckle_contrast

            contrast_array[r][c] = s_speckle_contrast
```

```
    conArr = np.asarray(contrast_array)
    conArr[conArr == 0] = conMin

    return (w_speckle_contrast, contrast_array)

def calcSpeckleContrastVal(image_path):
    x1 = 2150; x2 = 2600
    y1 = 1200; y2 = 1600

    image = plt.imread(image_path)
    intensity = np.asarray(image[y1:y2, x1:x2])

    stat_x1 = 0; stat_x2 = 50
    stat_y1 = 0; stat_y2= 50

    mov_x1 = 0; mov_x2 = 50
    mov_y1 = 200; mov_y2 = 250

    filtered = gaussian_filter(intensity, 2)

    static_window = np.asarray(filtered[stat_y1:stat_y2,stat_x1:stat_x2])
    stat_con = np.std(static_window) / np.mean(static_window)
    moving_window = np.asarray(filtered[mov_y1:mov_y2,mov_x1:mov_x2])
    move_con = np.std(moving_window) / np.mean(moving_window)

    return (move_con, stat_con)

def process_images(folder_path):
    results = {}

    # Iterate through the folder and process images
    for filename in os.listdir(folder_path):
        if filename.endswith('.bmp'):
            # Extract flowrate and exposure time from the filename
            parts = filename.split('_')
            flowrate = parts[1]  # Assuming flowrate is the first part
            exposure_time = parts[2]  # Assuming exposure time is the second part

            # Initialize the flowrate in results dictionary if not already
            if flowrate not in results:
                results[flowrate] = {}
```

```
            # Initialize the exposure time in the flowrate dictionary if not already
            if exposure_time not in results[flowrate]:
                results[flowrate][exposure_time] = []

            # Get the LSCI from the image
            move_con, stat_con = calcSpeckleContrastVal(os.path.join(folder_path, filename))

            results[flowrate][exposure_time].append(move_con)

    # Calculate the average for each exposure time within each flowrate
    averages = {}
    stds = {}
    for flowrate, exposure_times in results.items():
        averages[flowrate] = {}
        for exposure_time, values in exposure_times.items():
            averages[flowrate][exposure_time] = np.mean(values)

    # Convert results to DataFrame for saving
    df = pd.DataFrame.from_dict({(flowrate, exposure_time): avg
                                 for flowrate, exposure_times in averages.items()
                                 for exposure_time, avg in exposure_times.items()},
                                orient='index', columns=['Average Value']).reset_index()

    # Split the MultiIndex into separate columns
    df[['Flowrate', 'Exposure Time']] = pd.DataFrame(df['index'].tolist(), index=df.index)

    # Drop the original index column
    df.drop('index', axis=1, inplace=True)

    # Rearrange columns
    df = df[['Flowrate', 'Exposure Time', 'Average Value']]
    df.to_csv(folder_path + '/averages.csv', index=False)


# RUN CODE

folder_path = 'Data/0.001s'
process_images(folder_path)

#| label: tbl-planets
#| tbl-cap: Average K value for 0.001s Exposure Time
```

```
from IPython.display import Markdown
from tabulate import tabulate

# Display the csv as a Markdown table

table = pd.read_csv('Data/0.001s/averages.csv')
Markdown(tabulate(
  table,
  headers=["Flowrate [ml/hr]","Exposure Time [s]", "Average K [a.u.]"]
))
```

| | Flowrate [ml/hr] | Exposure Time [s] | Average K [a.u.] |
|---|---|---|---|
| 0 | 200ml | 0.001 | 0.365206 |
| 1 | 50ml | 0.001 | 0.367474 |
| 2 | 100ml | 0.001 | 0.332304 |
| 3 | 0ml | 0.001 | 0.365184 |
| 4 | 238ml | 0.001 | 0.359352 |
| 5 | 150ml | 0.001 | 0.371385 |

**Finding the LSCI Reproduction of Different Flowrates**

```
image_path1 = 'Data/0.001s/LSCI_0ml_0.001_1.bmp'
image_path2 = 'Data/0.001s/LSCI_50ml_0.001_1.bmp'
image_path3 = 'Data/0.001s/LSCI_100ml_0.001_1.bmp'
image_path4 = 'Data/0.001s/LSCI_150ml_0.001_1.bmp'
image_path5 = 'Data/0.001s/LSCI_200ml_0.001_1.bmp'

x1 = 2150; x2 = 2600
y1 = 1200; y2 = 1600


# Plot the Original Speckle Image
original = Image.open(image_path4).convert('L')
og_int = np.asarray(original)
og_int = og_int[y1:y2, x1:x2]

# Plot the speckle contrast

(cont1, contrast_array1) = calcSpeckleContrast(image_path1)
```

```
(cont2, contrast_array2) = calcSpeckleContrast(image_path2)
(cont3, contrast_array3) = calcSpeckleContrast(image_path3)
(cont4, contrast_array4) = calcSpeckleContrast(image_path4)
(cont5, contrast_array5) = calcSpeckleContrast(image_path5)

fig, ax = plt.subplots(nrows=2, ncols=3, figsize=(12, 4))

im0 = ax[0,0].imshow(og_int, cmap = 'gray', vmin=0, vmax=255)
ax[0,0].set_title("(a) Intensity of Laser Speckle for 150 ml/hr", fontsize = 8)
divider0 = make_axes_locatable(ax[0, 0])
cax0 = divider0.append_axes("right", size="5%", pad=0.05)
cbar0 = fig.colorbar(im0, cax=cax0)
cbar0.set_label('Intensity [a.u.]')

im1 = ax[0,1].imshow(contrast_array1, cmap = "jet")
ax[0,1].set_title(f'(b) Speckle Contrast of Laser Speckle for 0 ml/hr', fontsize = 8)
divider1 = make_axes_locatable(ax[0, 1])
cax1 = divider1.append_axes("right", size="5%", pad=0.05)
cbar1 = fig.colorbar(im1, cax=cax1)
cbar1.set_label('Speckle Contrast K [a.u.]')

im2 = ax[0,2].imshow(contrast_array2, cmap = "jet")
ax[0,2].set_title(f'(c) Speckle Contrast of Laser Speckle for 50 ml/hr', fontsize = 8)
divider2 = make_axes_locatable(ax[0, 2])
cax2 = divider2.append_axes("right", size="5%", pad=0.05)
cbar2 = fig.colorbar(im2, cax=cax2)
cbar2.set_label('Speckle Contrast K [a.u.]')

im3 = ax[1,0].imshow(contrast_array3, cmap = "jet")
ax[1,0].set_title(f'(c) Speckle Contrast of Laser Speckle for 100 ml/hr', fontsize = 8)
divider3 = make_axes_locatable(ax[1,0])
cax3 = divider3.append_axes("right", size="5%", pad=0.05)
cbar3 = fig.colorbar(im3, cax=cax3)
cbar3.set_label('Speckle Contrast K [a.u.]')

im4 = ax[1,1].imshow(contrast_array4, cmap = "jet")
ax[1,1].set_title(f'(e) Speckle Contrast of Laser Speckle for 150 ml/hr', fontsize = 8)
divider4 = make_axes_locatable(ax[1,1])
cax4 = divider4.append_axes("right", size="5%", pad=0.05)
cbar4 = fig.colorbar(im4, cax=cax4)
cbar4.set_label('Speckle Contrast K [a.u.]')
```

```
im5 = ax[1,2].imshow(contrast_array5, cmap = "jet")
ax[1,2].set_title(f'(f) Speckle Contrast of Laser Speckle for 200 ml/hr', fontsize = 8)
divider5 = make_axes_locatable(ax[1,2])
cax5 = divider5.append_axes("right", size="5%", pad=0.05)
cbar5 = fig.colorbar(im5, cax=cax5)
cbar5.set_label('Speckle Contrast K [a.u.]')

# Adjust layout
plt.tight_layout()

# Save the plot
plt.savefig('images/LSCIReal.png')
plt.close(fig)
```

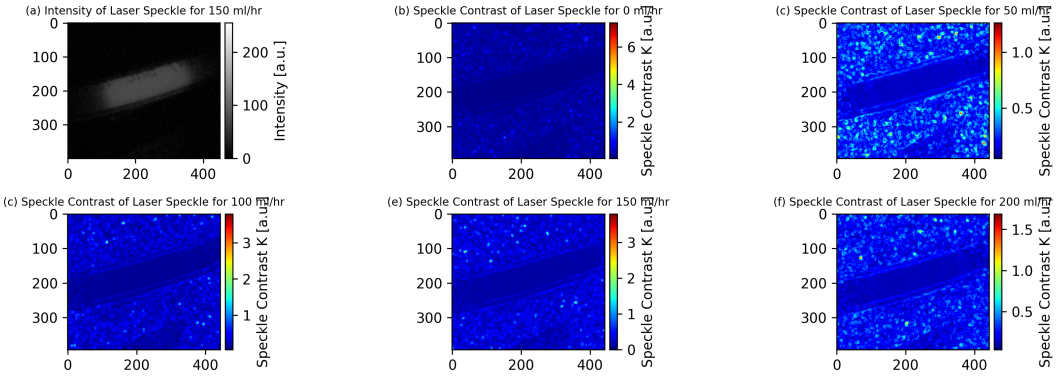


Figure 5.7: The real-world results for LSCI on a 150 ml/hr fluid.

## 5.4 Additional Capabilities

The system was successful in taking time lapse images over an extended period of time and could be used for a broad range of biological imaging applications. The time lapse of the Tobacco Hornworm growth resulted in 45 images over the span of 9 days, with 5 hours between each image. An example of one week's worth of growth is shown in Fig. 5.8, and an image of the full box on the first and last day is shown in Fig. 5.9.

While the OpenIVIS system was able to take images of lettuce leaf decomposition over a three day period, the anthocyanin response was not able to be determined. The response imaged on the 1st and 3rd days of the trial are shown in Fig. # and #. The anthocyanin response is very similar between the two images, and discernible changes are not able to be visualized. It is possible that the type of lettuce being imaged does not have a strong enough concentration

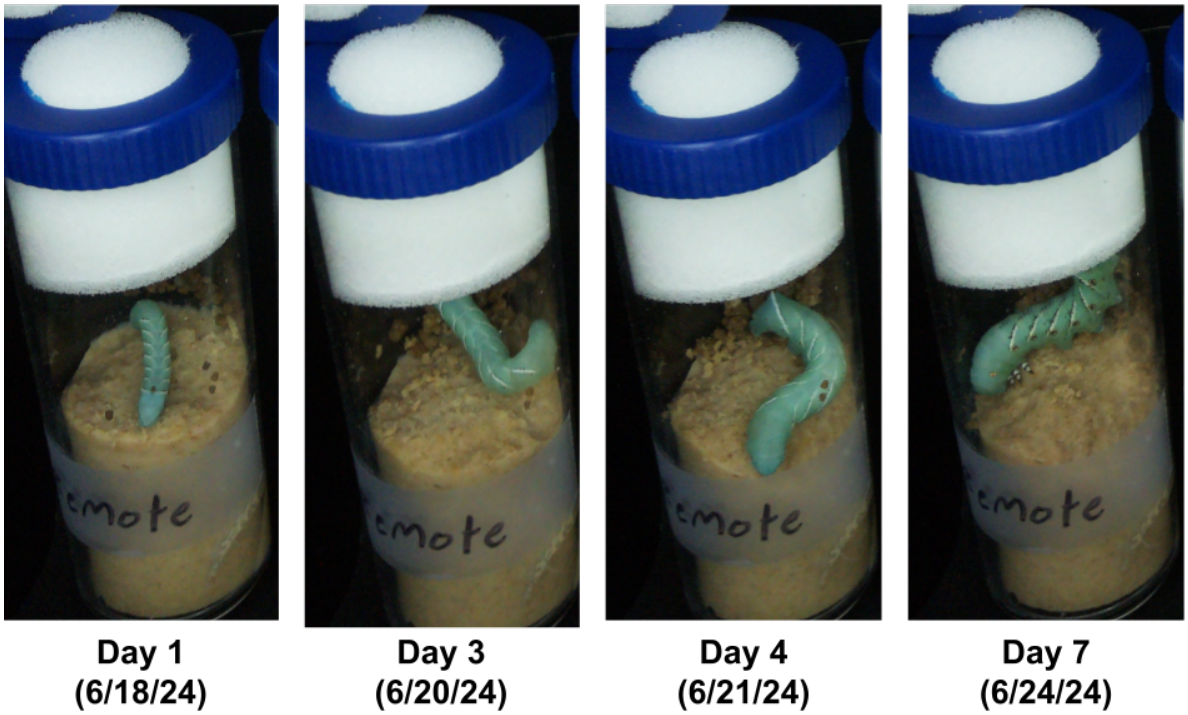


Figure 5.8: A week's worth of growth from one hornworm.

of anthocyanin to properly fluoresce, resulting in similar results. Additionally, due to limited documentation on how to conduct this experiment, the experimental process and calculations used may be incorrect. An accurate depiction of this trial is shown in the system's previous publication by Branning et. al [Branning].

# 6 Conclusion

# 7 Conclusion

The conclusion summaries the main results of your paper. It generally mirrors the abstract, but in slightly more detail.

It is also common for the conclusion to include a discussion of the data, limitations of the work, and directions for future work.

# 8 Acknowledgments

This research was supported by the Silicon Valley Community Foundation under Grant No. 2022-251322. The authors would like to thank the Colorado School of Mines Cash Lab, Harvey Mudd College, Daniel A. Guerra, Jenny Nguyen, Xavier Walter, and Jacob Staimpel for their support and collaboration.

- Daniel A. Guerra - HMC Chemistry Laboratory and Stockroom Manager
- Jenny Nguyen - HMC Biology Laboratory Manager
- Xavier Walter - HMC Staff Engineer
- Jacob Staimpel - HMC Engineering Instructional Support Coordinator

## 8.1 Funding Acknowledgement

A sample acknowledgement of funding by an NSF grant might read something like the following.

> This research was supported by the National Science Foundation under Grant No. [Your Grant Number]. The authors would like to thank [any other contributors, institutions, or facilities] for their support and collaboration. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the National Science Foundation."

[1] R. T. Williams and J. W. Bridges, "Fluorescence of solutions: A review," *Journal of clinical pathology*, vol. 17, no. 4, p. 371, 1964.

[2] G. Saleh, M. Faraji, R. Alizadeh, and A. Dalili, "A new explanation for the color variety of photons," in *MATEC web of conferences*, EDP Sciences, 2018, p. 01003.

[3] "Fluorescein (FITC) - US." Jul. 2024. Available: https://www.thermofisher.com/us/en/home/life-science/cell-analysis/fluorophores/fluorescein.html

[4] S.-Y. Lee and A. Ortega, "A novel approach of image compression in digital cameras with a bayer color filter array," in *Proceedings 2001 international conference on image processing (cat. No. 01CH37205)*, IEEE, 2001, pp. 482–485.

[5] S. W. Hasinoff, "Saturation (imaging)," in *Computer vision: A reference guide*, Springer, 2021, pp. 1107–1109.