Monitoring Early Stages of Bacterial Adhesion at Silica Surfaces through Image Analysis

by

Victor Sun
B.Sc., University of British Columbia, 2017

A Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of

MASTER OF SCIENCE

in the Department of Chemistry

Monitoring Early Stages of Bacterial Adhesion
at Silica Surfaces through Image Analysis

by

Victor Sun
B.Sc., University of British Columbia, 2017

Supervisory committee

Dr. Dennis K. Hore, Supervisor
(Department of Chemistry)

Dr. Alexandre Brolo, Departmental Member
(Department of Chemistry)

Supervisory committee

Dr. Dennis K. Hore, Supervisor
(Department of Chemistry)

Dr. Alexandre Brolo
(Department of Chemistry)

ABSTRACT

Understanding bacterial adhesion and biofilm formations on abiotic surfaces are important biological processes that affect the growth of bacteria, with its far-spreading impacts on in everyday life, either as a benefactor or as an inhibitor. To study these bacterial interactions, tools to probe these interfaces are also important to provide further means for discovery of the adhesion mechanisms. In this thesis, a customized imaging platform was developed, utilizing brightfield microscopy to study *E. coli* K12 on silica surfaces over the stages of bacteria growth. Results observed bacteria adhering onto silica surfaces in a preferential pattern to already existing bacteria-adhered colonies. This suggest that bacteria, once adhered to the surface, enhance attraction of other planktonic bacteria. The platform was designed to enable concurrent Raman measurements, with further optimization required in order to enhance the Raman signals from individual cells. Results from this study provide strong evidence to link changes in interfacial water structure with previous surface vibrational spectroscopy experiments, where the surface coverage of bacteria was found to reach a maximum earlier in the stages of growth compared to the surface water response, indicating that adhesion alone is not the primary contributing factor to modification of the surface microenvironment.

Contents

List of Figures

List of Symbols and Definitions

symbol	definition	units
E. coli	*Escherichia coli*	
EPS	extracellular polymeric substances	
LB	Luria-Bertani broth	
NA	numerical aperture	
OD600	optical density at 600 nm	
px	pixel	

ACKNOWLEDGEMENTS

I would like to thank my supervisor, Dr. Dennis Hore, for giving me the opportunity to work in his laboratory. Thank you for your knowledge, guidance, and patience; for instilling positivity and confidence in becoming an observant scientist. It was a wonderful experience being introduced to a new level of science and thinking.

I would also like to thank Dr. Stanislav Konorov, for his great insight and help with optics and spectroscopy.

Thank you to my fellow lab members for being a supportive group. Extra thanks to Tasha for all the continued discussions on our projects.

Special thanks to my dad for supporting me, getting me to where I am today. Yennie for the love and support.

Chapter 1

Introduction

1.1 Bacterial Adhesion

1.1.1 Early Cell Adhesion

Cells are known to adhere to and interact with their environment and with surrounding cells. Cell adhesion can be mediated by adhesive bonds between surface receptors and their cell ligands which are made up of various chemical and physical interactions [1]. Cells can also use their adhesive abilities to adapt to changes in their surroundings, allowing cell migration behaviour to vary, where this behaviour can be affected by various chemical and mechanical stimuli [1, 2]. The stages of bacterial adhesion can be summarized by five main stages: reversible attachment, irreversible attachment, biofilm formation, biofilm maturation, and dispersion [2].

Early bacterial adhesion begins when planktonic bacteria attach onto a surface. This adhesion is primarily mediated by non-specific interactions onto solid surfaces, or by specific ligand-receptor interactions [1,2]. The initial adhesion constitutes a readily bacteria reversibly attaching to a conditioned surface, dictated by various non-covalent interactions such as hydrophobic, electrostatic, and van der Waals interactions [1–3]. Since these are non-covalent interactions, the bacteria must be in close proximity to the surface for attachment to occur. Promotion of bacterial adhesion is also provided by cellular motility or fluidic flow until the bacterium reaches a critical enough distance to allow adhesion

to the surface to occur [1, 2]. Bacteria have receptors that help the initial contact, which includes pili and flagella to mediate motility onto the surface [1, 2]. This initial contact between cell-surface is also highly dependent on the surface conditions, as hydrophobic interactions having a strong influence on adhesion ability. Since bacteria, such as *E. coli*, have an overall negative surface charge to other negatively repulsive surfaces, such as silica, net repulsive forces must be overcome for adhesion to occur [1, 4].

In contact with water, Si-OH groups of silica dissociate to yield negative surface charge due to SiO^-, resulting in repulsive electrostatic forces between the surface and negatively-charged bacteria [4, 5]. In aqueous growth media, lipids, proteins, and other ions are able to condition the surface of substrates, allowing changes in electrostatic and hydrodynamic interactions, susceptible for bacteria to grow upon [6]. As bacteria adhere onto substrates, this causes change in the bacteria-substrate microenvironment in between the cell and the surface as they transport ions, secrete biological molecules; this would transform the microenvironment into something different than initially [6].

1.1.2 Chemical Stimuli for Bacterial Adhesion

To achieve irreversible adhesion onto surfaces, bacteria are known to have quorum sensing abilities, where they express specific genes to have a resulting phenotype that benefits the continued attraction of cellular attachment and a creation of biofilms [7, 8]. Biofilms also allow bacteria to begin to produce exopolysaccharides, a major component for future biofilms, between themselves and the surface. At this stage, other bacteria are able to stick together to begin to form aggregates of bacterial colonies [9]. Once securely attached the surface, bacteria begin to produce more mature biofilms, including extracellular polymeric substances (EPS), implicated in recruiting additional cells to the surface.

EPS consist of mostly polysaccharides, with smaller amounts of proteins, nucleic acids, and lipids. These highly dense biopolymers are secreted from bacteria during cell proliferation, with contributions from neighbouring lysed cells [10]. EPS have many

charged groups, which enhances the aggregation of bacteria onto surfaces. EPS can be categorized in two groups based on their distance from the cell surface, called bound and free EPS. The bound EPS are more tightly, covalently bound to the cell, whereas, free EPS are further away from bacteria and are more loosely linked as slime [11].

1.1.3 Biofilm Maturation

The growth of biofilms are favoured when the bacteria are in a healthy state of motility, where bacteria interactions, such as quorum sensing, leads to formations of larger, more three-dimensional structures [12]. However, biofilms are limited by the restricting availability of nutrients within their biofilms and in their microenvironment. Bacteria require sources of carbon, nitrogen, oxygen, proteins, lipids, and salts to continue to grow, where a lot of these components are contained within the media they live in [12]. As biofilms mature, and the bacteria grow and divide, nutrients from bulk media are slow to diffuse into the biofilms. This will cause changes in the survivability of bacteria due to the steric restrictions of the biofilm. At the microenvironment, changes in pH, ionic strength, oxygen, carbon, and other nutrients will occur, factoring into the viability of biofilms [12]. If these colonies and biofilms lack nutrients, bacteria will begin to change their behaviour from motility and adhesiveness to dormancy and survival, with the worst case being death. As biofilms mature, changes in nutrients, increases of toxic products will change the microenvironment between bacteria and surfaces. These changes would allow for the dispersion stage of bacterial growth to occur.

1.2 Adhesion Models

For early bacterial adhesion onto surfaces, previous studies suggest that bacteria have significant amount of biopolymers or EPS on their to surface to allow bacteria to adhere to the surface, overcoming the possible repulsive interactions between the cells and surfaces [13]. Studies also suggest that the media that bacteria grow in have contributing molecules

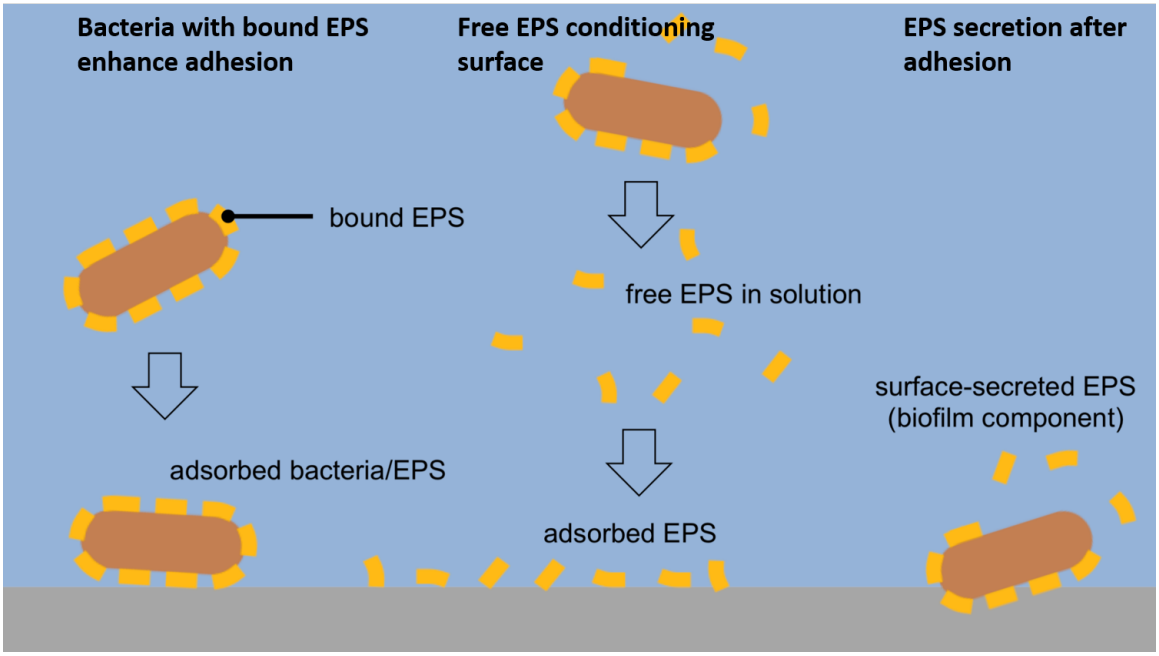

Figure 1.1: Proposed models of early bacterial adhesion involving EPS.

that condition the surface with ions and proteins that also allow bacteria to attractively adhere to the surface [6]. As seen in Figure 1.1, simplified models of bacterial adhesion are considered. In the first case, bound EPS surrounding the bacterium adhere to the surface. In the second case, free EPS that are floating and secreted from planktonic bacterium and used to condition the substrate before the bacterium adheres to the substrate. The third case, the bacterium adheres to the surface and then begins secreting EPS that attract other bacterium to adhere to the surface. These models have been investigated using nonlinear vibrational spectroscopy, where recently it was discovered that the interfacial water above silica substrate appears to be change over the stages of bacterial growth, suggesting that changes in the microenvironment may play a role in bacterial adhesion [14, 15].

The impact of bacterial growth on surfaces and these strongly adhering biofilms account for a large amount of microbial infections, contaminations, and biofouling, leading to increased rates of cost in medical and industrial settings. Many food contamination and food poisoning cases point towards excessive bacterial growth on products, which are motivated by optimal bacterial adhesion and large biofilm formations [16]. In medicine,

bacterial infections caused by biofilms growing on surgically implanted devices can prove to be fatal to the patient or otherwise be costly [17]. However, water treatment facilities are able to positively benefit from bacterial biofilm growth by using adhesive bacteria to filter water and remove unwanted matter [18]. By investigating and further understanding the complex mechanism of bacterial adhesion and biofilm formation, it will help lead investigations for further improved biomedical or biofouling applications.

1.3 Imaging Techniques

A large variety of techniques can be used to study bacterial adhesion and biofilm growth. Direct probing techniques include imaging and microscopy; adhesion strength techniques include atomic force microscopy (AFM), biomass determination, and chemical composition probing include cytometry and staining. [19]. Imaging techniques include traditional light microscopy such as darkfield and brightfield microscopy, a simple configuration utilizing a visible light source that is transmitted through or reflected on the sample and imaged onto a digital camera. More complex imaging techniques have been able to achieve higher-resolution, more detailed images such as AFM and fluorescence microscopy. AFM utilizes micro-sized, conducting metal tip illuminated with a laser to raster over a sample surface, to localize high resolution morphology information [20]. This achieves spatial resolutions in nanometer scales, however, is disadvantaged due to its complex design and fragile components, requiring long image acquisition times. Fluorescence microscopy also achieve high resolution as it utilizes fluorophores contained within a sample which are excited by an external light source. This detection of fluorophores localizes their position in a sample to a higher degree than simpler optical techniques. However, fluorescence microscopy may require long sample preparation times due to labelling samples with external fluorophores, and concerns with cell viability from photobleaching may be issues [21].

1.3.1 Brightfield Microscopy

For this thesis, the simpler configuration of brightfield microscopy is sufficient for morphology studies of *E. coli*. As brightfield microscopy allows samples or objects to appear dark on a bright background, it provides fast, reliable imaging results at the microscopic level, enough to see single cell sizes of *E. coli*, while keeping sample preparation at its lowest. However, disadvantages of this method is the loss of information if details lower than single cell sizes are required. As well, no chemical information can be obtained from this method, unless it is coupled with spectroscopy methods such as Raman spectroscopy.

1.3.2 Image Segmentation

In this thesis, image processing is used as a quantitative method to acquire bacterial adhesion coverage information from images. Image segmentation is used to simplify the image to define and label every pixel in the image as either bacteria foreground or empty background [22]. There are many image segmentation methods, however, common methods include edge detection, watersheding, and thresholding.

Edge detection methods are based on analyzing the first derivative of pixel intensity spatially, the changing of pixel intensity of neighbouring pixels in an image. By identifying the greatest areas of change, the boundaries of specific features of an image can be detected and connected together to form complete object boundaries. The advantage of this method is that highly detailed segmentation due to sensitivity of the detection of the object's edges, however, this method is prone to either under- or over-detecting these edges unless specific parameters were in place. For microscopic studies of bacteria that require accurate representation of all the cells within an image, this may be undesirable [22].

Watershed based methods utilizes converting images to three-dimensional topological surfaces based on pixel intensity and identifying ridge lines [22]. This results in more continuous, detected edges of foreground, and ridges are possible to be detected even with

small changes in pixel intensity, or lower contrast images, however, there are complex calculations and prepossessing of the image to determine the gradient of topography that are considered [23].

1.3.3 Quantification of Bacteria Using Image Thresholding

Image thresholding is utilized in this thesis as the method of segmentation. This method uses a simple, global threshold cutoff value to divide the image pixels relative to their pixel intensity, turning gray-scale images into binary images [24]. The then binary image can be processed for further analysis [22, 25]. The advantage of using thresholding as the image processing technique is that it is simple and fast, requiring very little parameters to achieve a desirable image segmentation. The disadvantage of this method is that globally thresholding an image does not discriminate in spatially detail regions, as thresholding cutoff is applied uniformly on the image, however, as the images to processed are of single, similar bacteria, this image segregation technique should be sufficient [22, 25].

1.3.4 Customization of Imaging Platforms

Commercial imaging platforms are robust, however, add-ons that increase their capability may come with physical limitations (reduced flexibility to add custom optics) and increased cost. With lab-built systems, many advantages allow for continued customization (additional modifications and upgrades), simple image acquisition set-up, automated image and stage control that can be tailored to specific research needs. With this flexibility for hardware, software can also be custom developed as well.

1.4 Objectives

Although the stages of bacterial adhesion is summarily known, more detailed biophysical mechanisms of the initiation of bacterial adhesion onto abiotic surfaces are lacking. When forming initial micro-colonies, bacteria produce EPS that provide stability to the

growth of bacteria and biofilms onto either other bacteria or substrates. Investigating cell-surface interactions and microenvironment in the context of bacterial adhesion could have significant impact in appropriate applications. Recent work from the group showed from nonlinear vibrational spectroscopy probing the interface above the silica substrate to have an increase in water orientation over the time of bacterial growth stages, suggesting that changes in the bacterial-substrate microenvironment may play a role in the adhesion or biofilm growth.

This thesis seeks to directly investigate this early irreversible bacterial adhesion on silica surface through imaging. The first objective is to develop an imaging platform to allow for automated, mass-throughput of brightfield imaging. The second objective is to investigate initial *E. coli* K12 adhesion onto silica surfaces through imaging techniques and analysis, to propose connections between the early stages of bacterial adhesion and the bacteria-silica microenvironment. The third objective is complement imaging with Raman spectroscopy to acquire specific chemical biomarkers that could contribute to bacterial adhesion.

Chapter 2

Methods and System Development

2.1 Brightfield Microscope Development

This thesis utilized a lab-built microscope together with custom software for instrument control and data analysis. The designed capabilities of the system were to be able to automate the acquisition of images at specific positions and to collect simultaneous Raman spectra. Compared to a commercial microscope, the system also offers flexibility for introducing additional imaging modalities (e.g. polarization, dark field, phase contrast) at a later stage. Microscopy was used as an imaging system to provide morphology information to identify bacterial adhesion on surfaces.

2.1.1 Microscope Hardware

A brightfield microscope system was employed to acquire images with a magnification sufficient to identify *E. coli*. A simplified schematic of the system is shown in Figure 2.1. It consisted of a collimated 6500 K colour temperature whitelight from an LED illuminator (MCWHL5, Thorlabs), focused onto the sample using various plano- and bi-convex lenses, and collected using a 50× NA 0.55 long-working objective lens (Plan L, Nikon). The focused whitelight beam was transmitted through the sample, and directed to the the high-magnification path into a CMOS camera (DCC1240C, Thorlabs). A second image collection pathway was implemented for low-magnification imaging, however, was left unused for the experiments described in this thesis.

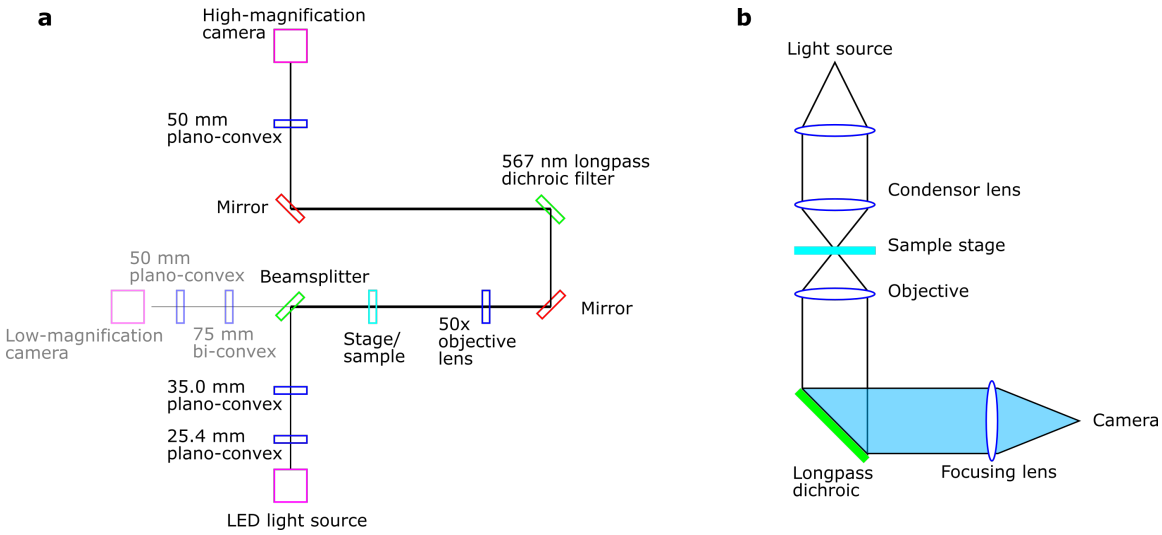

Figure 2.1: (a) Scheme of the main components of the brightfield microscope for ex vivo bacterial imaging studies. (b) Simplified high-magnification configuration of transmitted brightfield imaging

The microscope was equipped with three motorized stages (MTS25/KDC101, Thorlabs), allowing for programmable control of the sample positioning and focus. A custom laser-cut platform was used as the sample stage, capable of holding a 75 mm $\times$ 25 mm microscope slide. Due to the customization of the sample stage, other stages can be created to hold flow chambers, coverslips, other samples; only restricted by the range of the motors: positional range of 25 mm in the x- and z-positions, and 11 mm in the y-position. The completed construction of the microscope can be seen in Figure 2.2.

2.1.2 Brightfield Imaging

The optimization and calibration of the brightfield imaging system was done using a sample of stained crystal violet *E. coli* on a microscope slide. The field of view of the high-magnification image was calculated to be 210×160 μm. The microscope was used in the high-magnification configuration as a transmitted brightfield microscope, as seen in Figure 2.1(b). A 567 nm longpass filter and irises were added during the sample collection, to reduce chromatic aberrations, resulting in a largely blue-hue image, as seen in Figure 2.3.

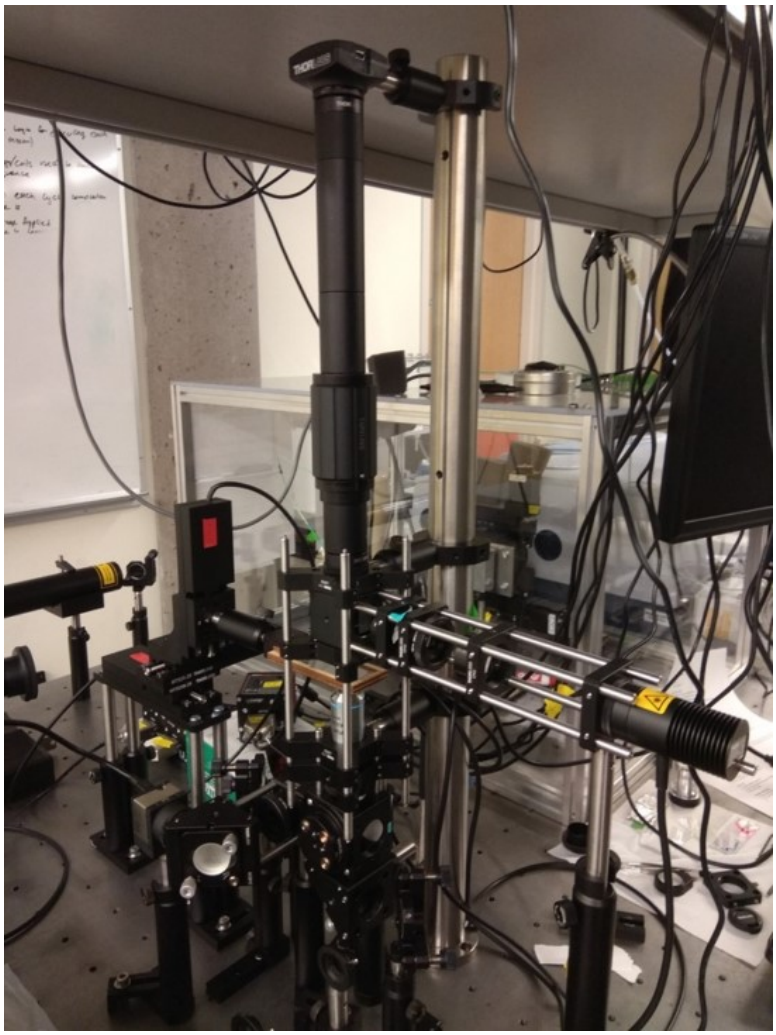

Figure 2.2: Completed construction of the microscope.

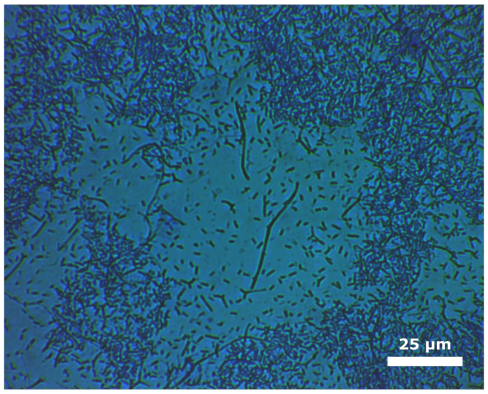

Figure 2.3: A sample of a brightfield image of *E. coli* on glass microscope slide, used for optimization of the imaging system. Bacteria were cultured in LB-enriched water for 8 h before being crystal violet stained.

2.1.3 Microscope Control Software

All microscope control and image analysis was written using Python, sample and software code shown in Appendix A. The system runs under the Linux operating system, allowing for flexibility and customization. Linux is also a robust system that is less prone to bloatware, is long-lasting, runs on older hardware, and offers many features for multi-user environments, fine-grained network access, and security.

Programmable microscope control involved developing a package consisting of a collection of scripts to have control of both the sample stage and camera for data acquisition. Scripts were created for stage control, modifying an open-source module for ThorLab's APT motion controllers, the primary motors that controlled the sample stage [26]. The initial open-source module was capable of allowing control for MTS50 stages with TDC001 motor controllers, however, the module was modified to allow control for the sample stage's MTS25 stages with KDC101 motor controllers. With complete control of the sample stage, the primary functions were to design automated scripts to move the sample stages in specific stepsizes or absolute positions on a sample.

Image acquisition was achieved by utilizing an open-source module derived from the

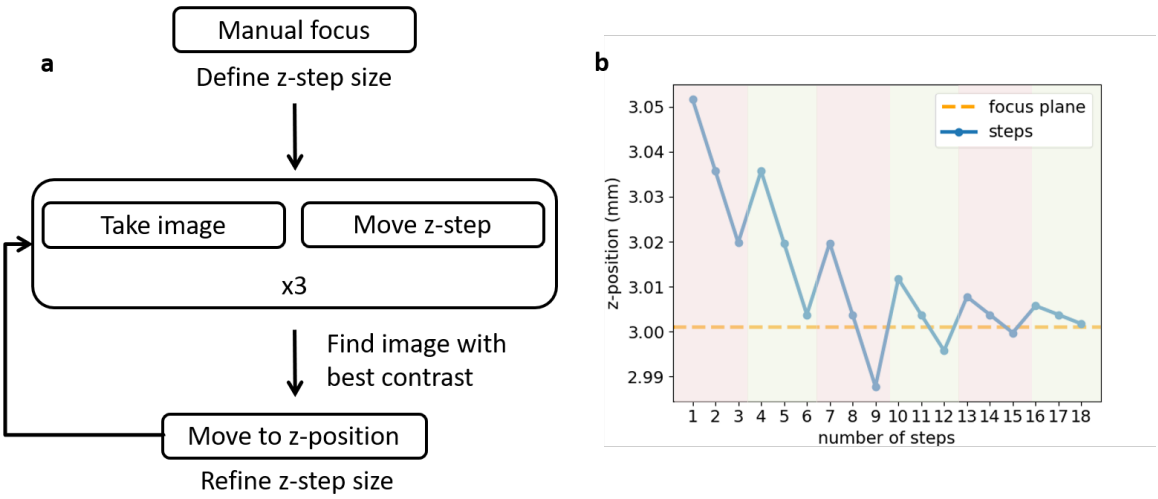

Figure 2.4: (a) Scheme of the autofocusing algorithm. (b) Observed z-position vs. time as the microscope automatically focuses.

software development kit for IDS cameras, the developer for the CMOS camera used in the lab-built microscope [27]. Camera control allowed for control over the acquisition time and exposure time of images. A complete integration of both stage and camera control through scripts allowed for custom functionality on the microscope, including autofocusing and tiling images.

Autofocusing Algorithm. Although the sample stage was designed to have a flat surface for clear imaging for focused samples, changes in stage position, in millimeter ranges, caused the images to defocus in the z-plane, requiring an autofocusing capability for the microscope. The algorithm utilized acquiring images at initial 16 μm step sizes, at three z-positions, moving the stage to the position with the image with the best contrast, reducing the step size, and repeating until the best contrast is observed within a z-spatial resolution of 1 μm. The generalized autofocusing scheme can be seen in Figure 2.4. This method worked well, with an initial coarse manual focus made visually before allowing the autofocusing to complete the fine adjustment.

The contrast was quantified by analyzing the pixel intensity distribution of the image. All images are assumed to have a bimodal intensity distribution due to having a uniform bright light of the background, and an ideal uniform darker foreground of the morphology

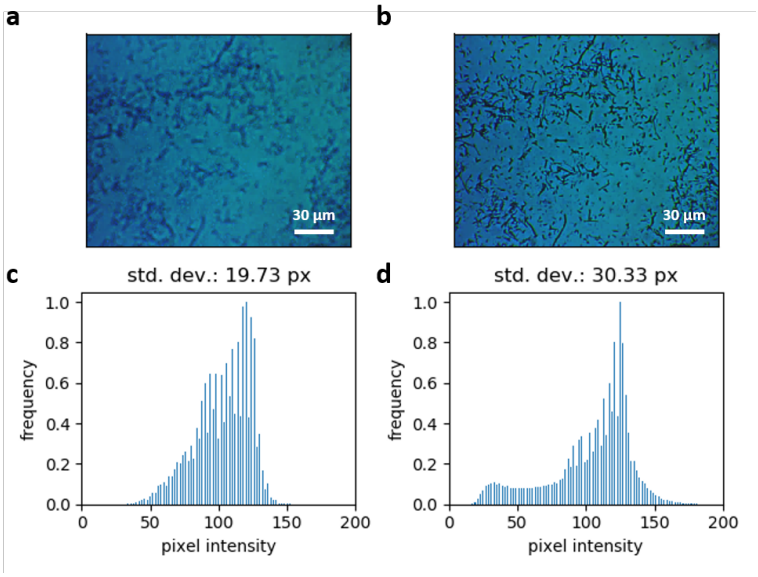

Figure 2.5: (a, b) Unfocused and focused image of *E. coli* at two different z-planes. (c, d) Pixel intensity distribution of the above images, with their resulting standard deviation.

of the sample. As unfocused images are blurry, it loses the sharp darkness of the foreground and blends more with the lighter background, having a more unimodal intensity distribution of pixels. By running a simple standard deviation of the pixel distribution, it gives a fast, sufficient indication of contrast of an image. By comparing multiple images, the image with the highest standard deviation, suggesting a less unimodal distribution, can be identified as an image with better contrast. Figure 2.5 shows the same sample at two different z-positions, with their pixel intensity distribution and standard deviation result. Comparing the two focus planes, the unfocused image yielded a more uniform distribution with lower standard deviation.

Pattern Rastering. To allow for large areas of the surface to be imaged, spatial pattern rastering, as seen in Figure 2.6, was developed to acquire large tiled areas of the surface. This capability allowed imaging at specific position on a sample surface given a specific step size and/or range of area. The pattern rastering is capable of stitching side-by-side image with overlapping field of views, however, this was not used in this thesis; by avoiding stitching overlapping field of views and taking points within an area of a sample greatly

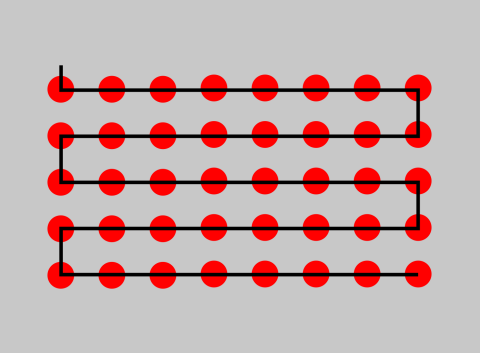

Figure 2.6: Scheme of the rastering pattern used for tiling large, millimeter areas, of the sample.

reduced the data acquisition time by decreasing the amount of images required, and allowed for greater area of representation of a sample surfaces for analysis.

2.2 Imaging Analysis

Custom image analysis scripts were developed for processing the data. Image thresholding allowed for distinct segregation of foreground and background particles, indicative of adhered bacteria and the substrate surface. Information from image thresholding further allowed detailed analysis on foreground and bacteria, including bacterial surface coverage, and single cell and single colony sizes. A scheme of the image analysis is shown in Figure 2.7, which is summarized in three main steps: the initial image thresholding, bundling data from multiple images, and post-thresholding analysis.

2.2.1 Image Thresholding

Analysis was done by image thresholding based on a modification to the Otsu's method [28], generating bacterial coverage and colony size information relative to the area of imaged surface. As seen in Figure 2.8, Otsu's method utilized an algorithm that can determine a optimal threshold, t, where the variability of the foreground and background pixel level intensities are optimized to transform an image from grayscale to binary while

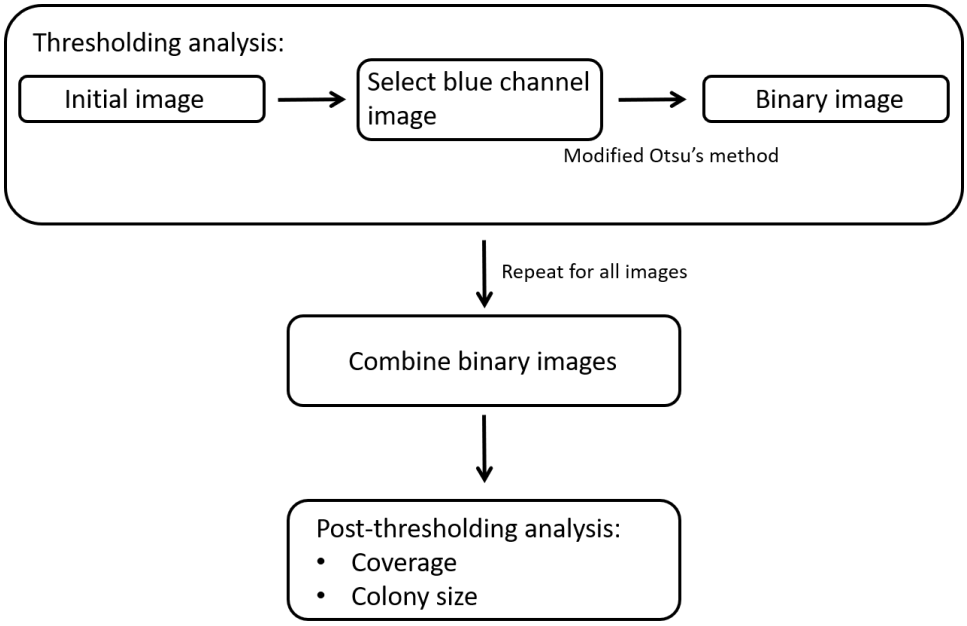

Figure 2.7: Simplified scheme of imaging analysis done for bacterial studies.

retaining foreground pixels as one value, and background pixels as another value [29].

$$\sigma_w^2(t) = \omega_0(t)\sigma_0^2(t) + \omega_1(t)\sigma_1^2 \tag{2.1}$$

Equation (2.1), with σ being the variance of the pixel intensities, ω being the summation of the pixel intensities for one region, is minimized to determine t, over the complete range of possible thresholds, where $\omega(t)$ are the sum of the probabilities between one of the two bimodal distributions, as seen in Equation (2.2) and Equation (2.3).

$$\omega_0(t) = \sum_{i=1}^{t-1} p(i) \tag{2.2}$$

$$\omega_1(t) = \sum_{i=t}^{L} p(i) \tag{2.3}$$

Otsu's method performed well if the image histogram exhibited a bimodal distribution, a background of specific light intensity, contrasted with a dark foreground of stained bacteria. However, Otsu's method suffered from images with a weak bimodal distribution, or extreme cases of little or too much foreground, requiring a modification to the algorithm. The modified Otsu's method determined the ratio of sum of the probabilities between two

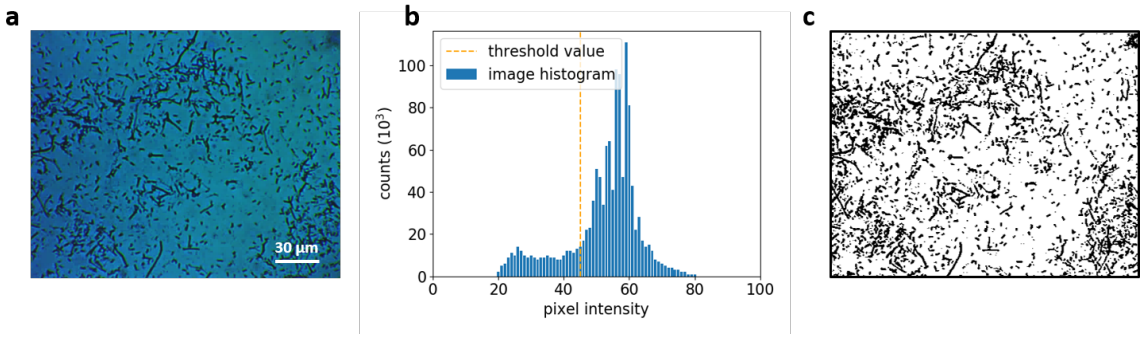

Figure 2.8: (a) Original *E. coli* image. (b) Image histogram with modified Otsu's method determined threshold value. (c) Binary image after thresholding.

bimodal distributions at $L/2$ threshold, where L is the maximum pixel intensity of an image. As the ratio of the sum of the distribution at $L/2$ deviated from 50%, a parabolic function was applied to the distribution, giving the Otsu's method a stronger threshold value when further away from the ideal bimodal distribution, to identify the true foreground and background at extreme cases of non-bimodal distributions, as seen in Equation (2.4) and Equation (2.5).

$$w_{L/2} = \sum_{i=1}^{L/2} p(i) \tag{2.4}$$

$$T = w_{L/2}^{10} t \tag{2.5}$$

The performance of the modified thresholding analysis was compared to the default Otsu's method, using simulated images with a known percent coverage of foreground, identified as simulated *E. coli*, as seen in Figure 2.9. The unmodified (original) Otsu's method performed poorly in extreme cases of too little and too much foreground, where low amount of simulated *E. coli* allowed for misclassifying portions of the background as the foreground coverage, increasing the expected coverage; whereas, high amount of simulated *E. coli* allowed for misclassifying portions of the foreground as background. The modified Otsu's method successfully mediated the extreme cases, however, in general, Otsu's method identified the foreground coverage as slightly lower than its true value.

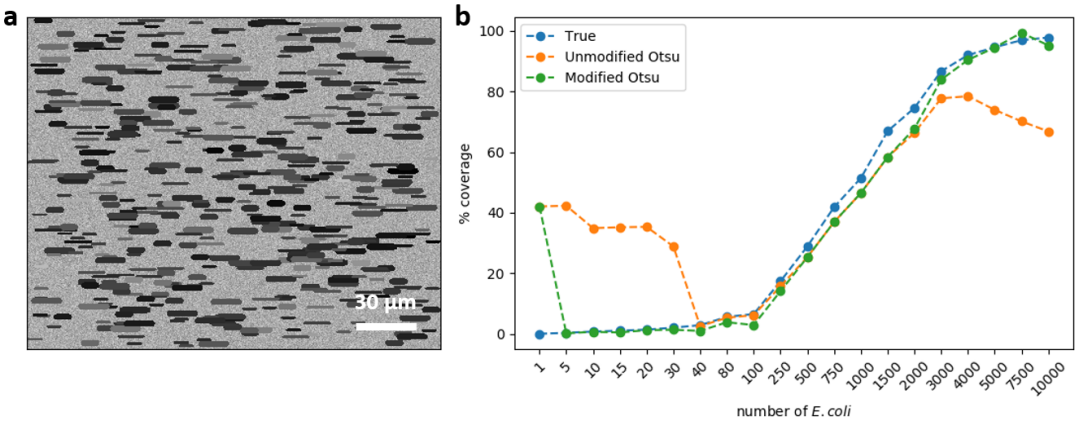

Figure 2.9: (a) Simulated image with randomly distributed rod-shaped foreground particles, used as verifiable foreground. (b) Comparison of Otsu's method, modified Otsu's method, and true foreground coverage from one simulation.

2.2.2 Post-thresholding Analysis

Proper image thresholding thus allowed for analysis of *E. coli* at the silica surface in terms of percent coverage, colony size, and single bacteria size measurements.

Percent Coverage. Percent coverage relative to the total area of the image was calculated based on the pixels foreground vs. background determined from the thresholding analysis.

Colony Size. Identification of colonies were done by a function to identify any bordering foreground pixels within an image as one defined colony, where a single cluster of bacteria are defined as foreground pixels that have no bordering pixels. The advantage of this identification allowed for faster processing time and does not discriminate between different shapes of *E. coli* (at their different orientations of attachment or stages of growth), however, the disadvantage is that it cannot differentiate between two bordering colonies or cells, therefore would consider them as a single cluster.

Single Cell Size. Single cells were identified by restricting the amount of foreground pixels of of the images to sizes equivalent to 0.5–4.0 μm^2, a boundary that would allow single *E. coli* with known cell sizes of 0.7–2 μm $\times$ 0.7–2 μm in size to be identified, while restricting unwanted cell colonies or small artifacts. For each time point, 100–400 single

cells were identified for the size calculation, with the mean and standard deviation being calculated with three experiments.

2.3 Experimental Procedure

2.3.1 Growth Conditions

E. coli K12 MG1655 (ATCC 700926) were used in all experiments. *E. coli* has been widely studied, with extensive understanding of their bacterial adhesion mechanisms on variety of substrates [30]. *E. coli* is a Gram-negative, rod-shaped bacterium, containing peritrichous flagella for motility. The size of a single cell varies from 2–3 μm $\times$ 0.25–1.0 μm [31, 32].

E. coli K12 MG1655 is useful as an experimental species as it can be easily grown in a laboratory setting, having a short reproduction time of twenty minutes, allowing for fast growth onto surfaces within 24 h [33]. Although the specific strain of MG1655 produces less biofilm, it is a sufficient strain to study bacterial adhesion at its early stages [34].

The bacteria were maintained at -80°C in 1% glycerol-enriched LB (Lennox, BD Difco) media for long-term storage. For regular use, bacteria were grown on LB agar plates, maintained at 4°C. In preparation for experiments, *E. coli* K12 were subcultured in LB media for approximately 20 h, until the optical density at 600 nm (OD600) of the solution reached 1.0. A sample of the subculture was subsequently diluted 1:100 by volume with fresh LB and incubated aerobically in a 37°C shaking incubator, at 180 rpm, over 24 h inside a custom-built glass coverslip setup. This setup allowed for bacterial adhesion to occur on upright silica glass coverslips.

Figure 2.10 was the design of the setup, made from polylactic acid on a 3D printer (Ultimaker 3). The final design was capable of holding 16 coverslips in the upright position, fitted into a custom bevelled 800 mL beaker for cell growth experiments. To ensure enough imaging replicates and temporal resolution, two holders were ran per experiment. The silica glass used were 22 mm $\times$30 mm No. 1.5 coverslips (ThermoFisher).

Figure 2.10: Design of glass coverslip holder used for bacterial experiments.

2.3.2 Coverslip Imaging

Bulk cell population growth was monitored by OD600, a light scattering method indicative of the density or concentration of live cells within the bulk media. The coverslips, in replicates of four, were removed at specific time points. To remove unadhered *E. coli* and to improve the imaging quality, the coverslip samples were dipped rinsed with deionized water, gram stained with crystal violet (BD BBL), and dried. Gram staining enabled higher contrast in imaging as a biomass indicator [35]. The image thresholding procedure (the modified Otsu's method) enabled quantification of the adhered bacteria on the surface using a simple stain such a crystal violet.

The sample coverslip was laid on a glass microscope slide, and inserted on the sample stage. The sample was arranged on the stage to ensure the images were captured in the middle of the coverslip. Every coverslip was imaged using the spatial pattern rastering to acquire 40 (5×8) different points in 1 mm $\times$ 2 mm steps, within an 8 mm $\times$ 16 mm area at the centre of the coverslip. These parameters allowed a representative image of the silica surface, while maintaining efficient time scales for data analysis.

Chapter 3

Bacterial Adhesion Study

3.1 Cell Growth and Imaging

The concentration of *E. coli* population was monitored using the optical density at 600 nm (OD600), as seen in Figure 3.1. The population growth rate followed the expected logarithmic growth rate model. The three main phases of growth were observed; the lag phase from 0-4 h, exponential phase from 4–8 h, and stationary phase from 8–24 h. At each time point observed, cover slips were removed from the growth solution, discontinuing adhesion, and imaged using the microscope. Figure 3.2 showcased a subset of 25 images taken from one removed coverslip. Due to the heterogeneity of bacterial coverage on the surface, a large set of images were required to enable a representative sample of bacterial coverage on silica surface.

During the various stages of cell growth, single cell size may vary due to changes in their surroundings, including nutrient changes and mechanical stress [36]. This was observed, as cell size over the duration of the experiment were visibly changing. This change in single cell size may also have an overall impact on further analysis on coverage, therefore, determining the change in single cell size over time was required. As seen in Figure 3.3, during lag and stationary phases, cell size was calculated to be approximately 2 μm^2 in size, while during the exponential phase, the most rapidly growing phase, had a slightly visibly larger cell size. It is evident that at stationary phases of growth, cell size shrunk, suggesting changes in the bacteria's surroundings may be a cause. Over time,

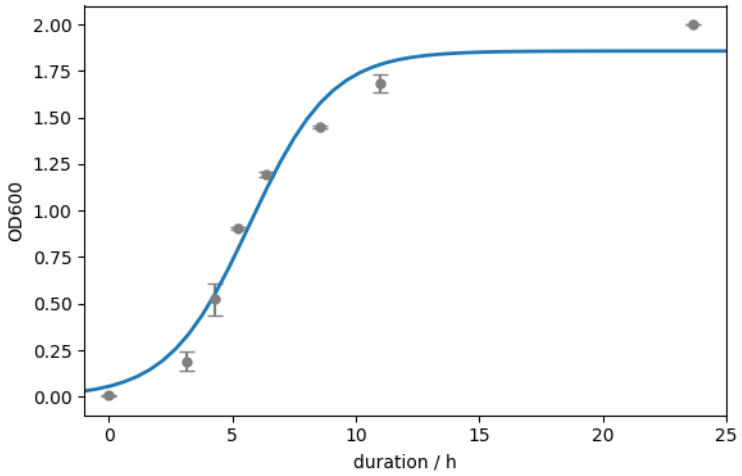

Figure 3.1: Mean and standard deviation of bulk cell culture growth curve determined by OD600 from three experiments with $N = 3$.

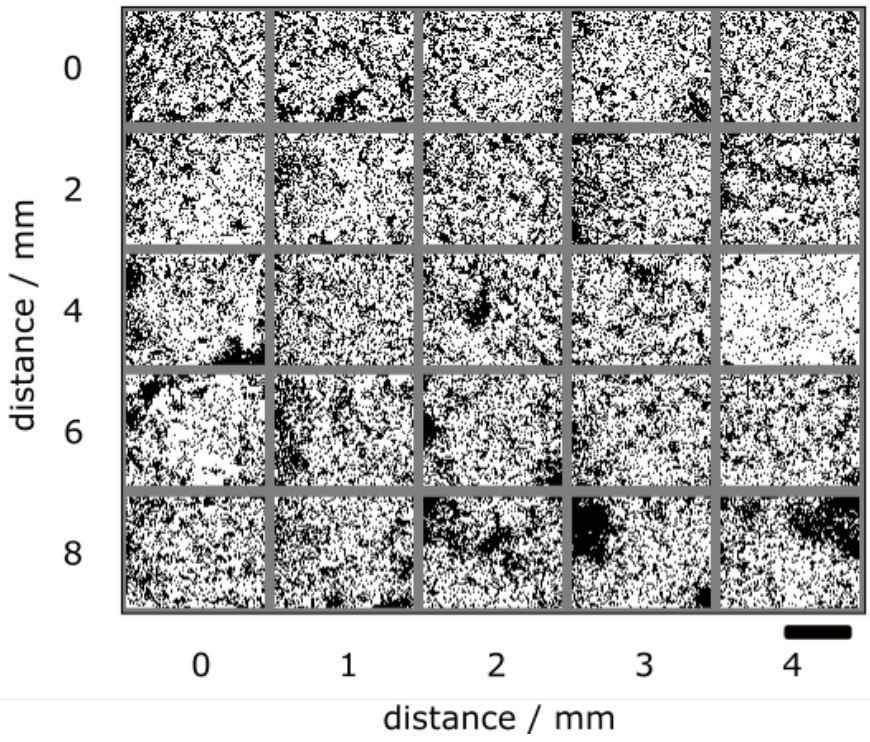

Figure 3.2: Subset of images taken from one coverslip during the stationary phase of growth. 25 binary images, after thresholding. The scale bar has a length of 100 μm.

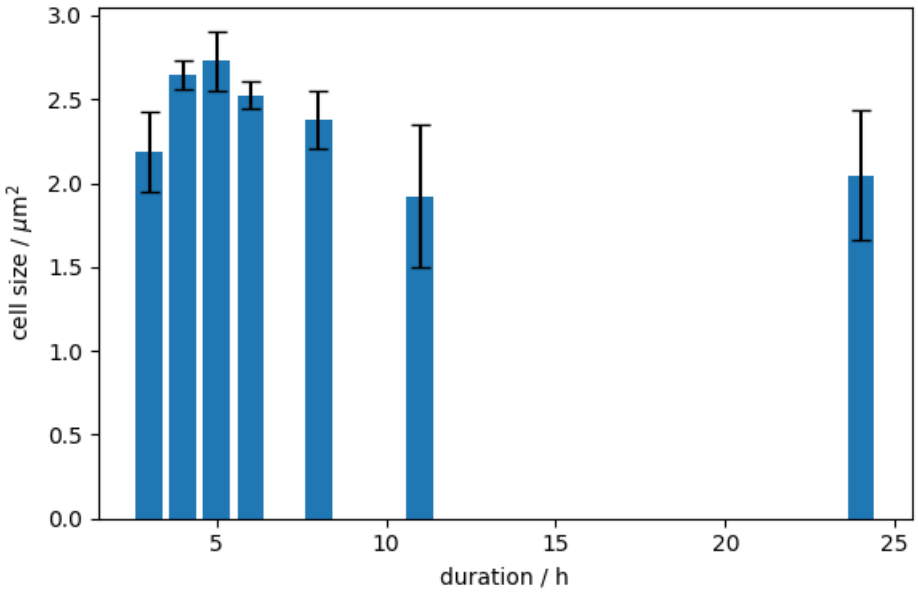

Figure 3.3: Mean and standard deviation of single bacterium size *vs.* duration in three experiments. Error bars represent one standard deviation about the mean.

nutrients in the growth media would get depleted and bacteria would starve [37]. Facing starvation, bacteria could change their motility behaviour, and their cell size [38].

3.2 Surface Coverage Analysis

Image analysis of the bacteria adhered silica surfaces yielded the amount of bacterial coverage relative to the surface of silica over time, as seen in Figure 3.4. The temporal trend of coverage appeared to resemble exponential adhesion of cells during the lag and exponential phases. however, bacterial coverage dropped during the stationary phase of growth from 50% to 30%. One cause of this dip in coverage may be due to the changing cell sizes over time, however, the drop of coverage cannot be completely corrected with the change of single cell size. One other factor that may explain the loss of coverage may be during stationary phase of growth, cells no longer prefer to adhere to surfaces, changing their behaviour, and due to the continued turbulent, shear forces during growth, cells would also begin to dispersing from the surface [38].

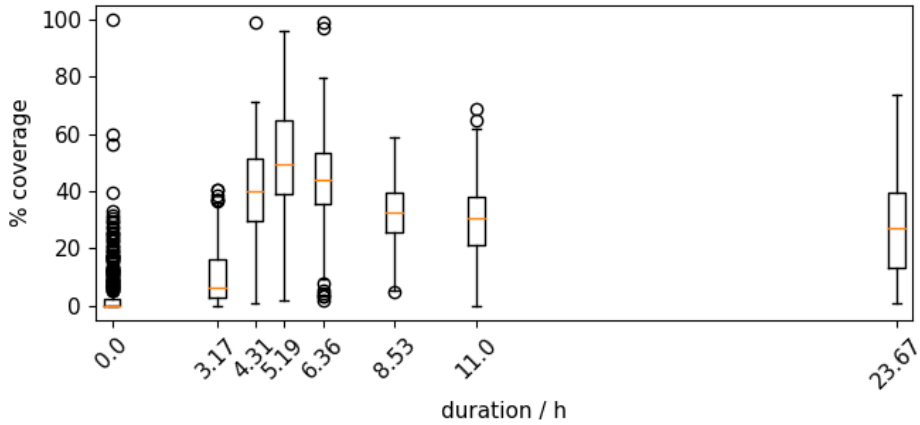

Figure 3.4: Boxplot of percent coverage vs. duration in three sample experiments. Percent coverage values with the median shown in orange, the box shows the interquartile range, and the whiskers showing the maximum and minimum coverage calculated for the time point. The circle dots represent outliers that were not used for the calculation of the boxplot. Each time point consist of $N = 480$ images.

To ensure the image thresholding was not misclassifying other artifacts as foreground, and therefore misidentified bacteria, modification of Otsu's method was done to ensure extreme cases of very low and very high foreground could be appropriately adjusted. For cell growth experiments, the 0 h time point for cell coverage contained zero bacteria and should result in zero surface coverage and zero colony size information. Experimentally, the 0 h time point yielded percent coverage of 4%, suggesting the image classification algorithm may not be performing perfectly, however, was sufficient for this study. Reasons for this misclassification included residue crystal violet stains on the silica surface, or uneven illumination acquired on the image. However, images in later stages of growth visually contained less staining residue, allowing for Otsu's method to perform effectively.

As variability is evident in biological studies, it is important to acknowledge the variability and not only represent averaged data. We confirmed the variability of bacterial coverage over three sample experiments, with four cover slip replicates per time point, keeping all parameters constant. In Figures 3.5 and 3.6, histograms of surface coverage

and colony size per time point per image are shown. The boxplot, as seen in Figure 3.4, represents 480 images' percent coverage values with the median shown in orange, the box showing the interquartile range, and the whiskers showing the maximum and minimum coverage calculated for the time point. The circle dots represent outliers that were not used for the calculation of the boxplot. This suggested that large data sets of images are required to acquire statistically significant information from these experiments, which is why the experiments utilized the spatial pattern rastering, acquiring an upwards of 480 images per time point shown.

3.3 Cell Colony Analysis

Although cell coverage analysis may show the overall cell population adhesion behaviour on the surface, it does not discriminate between small, freshly adhered cells, compared to larger forming cell colonies. Figure 3.6 showed the histogram of cell colony sizes determined, where the growth experiments observed median colony sizes remained around 4 μm^2 over time. However, averages may not represent the cell colonies sufficiently, as it is possible that large cell colonies contribute more to surface coverage than the small colonies. Figure 3.7 analyzed the categorization of two groups, a small group of cells below 100 μm^2, and a larger group of cells above 100 μm^2. Over time, it is shown that the number of small colonies are adhering at a much faster rate than the number of large colonies, however, the surface coverage is mostly dominated by larger colonies until stationary phase, as seen in Figure 3.7(c) and (d).

3.4 Cell Adhesion at the Population Level

Summarized in Figure 3.7(a) and (b), the rate of the cell surface coverage occurs the fastest during lag to mid-exponential phase of bulk cell growth, suggesting that this was the most optimal time for bacterial adhesion. During late-exponential to stationary phase, the surface coverage decreases, suggesting cells are being dispersed from the surface. However, during

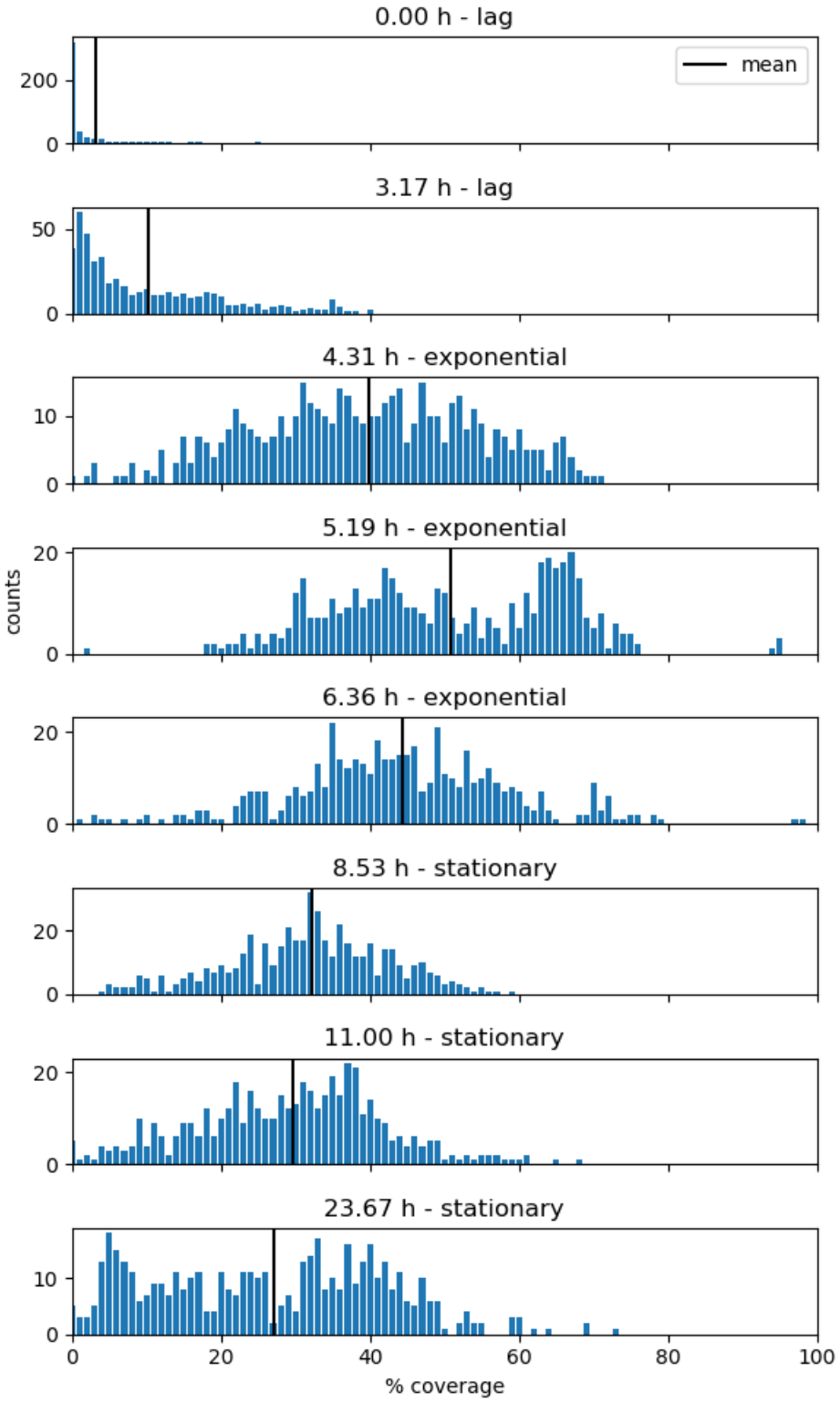

Figure 3.5: Histogram of cell coverage for three separate sample runs of $N = 480$ images per time point.

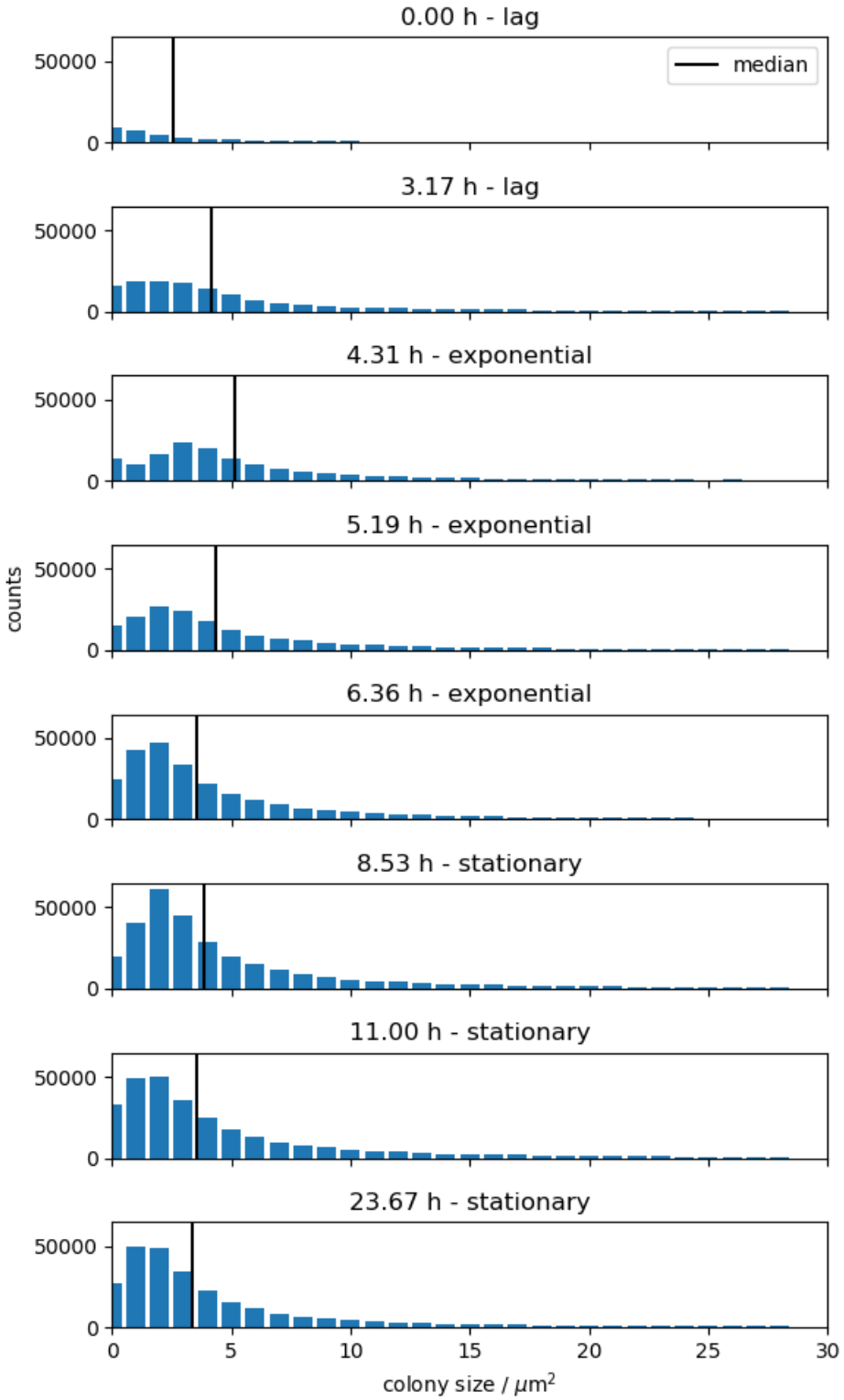

Figure 3.6: Histogram of cell colony size of combined three separate sample runs from $N = 480$ images per time point.

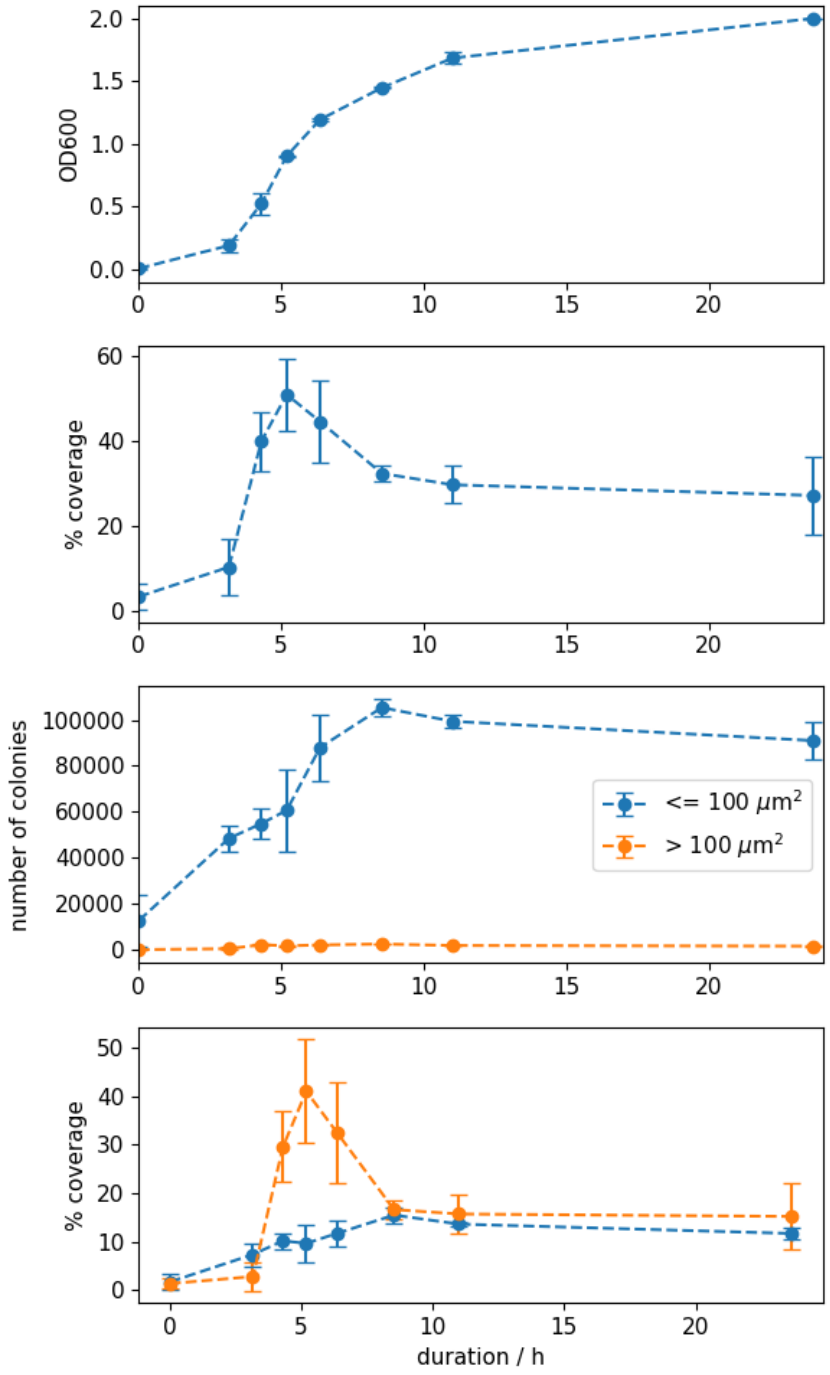

Figure 3.7: Categorizing small and large colonies based on a threshold of 100 μm^2. (a) OD600 growth curve. (b) Mean percent coverage of all colonies. (c) Number of colonies of categorized size. (d) Mean percent coverage of categorized size. Errorbars are of the standard deviation of the means of three experiments.

this time, the number of cell colonies continue to trend upward or remain constant onto the stationary phase, as seen in Figure 3.7(c). This suggest that only large colonies are being dispersing from the surface, while maintaining small colonies. This is also verified by analyzing the categorized large colonies, the larger subset of cell colonies contribute to the surface coverage the most during lag to mid-exponential phase, up to 40% of the fraction (50%) of the population at mid-exponential phase. However, by stationary phase, large colonies contribute 20% (of the 35%) of stationary phase surface coverage. Looking at bacterial adhesion at this population level is beneficial to give a general sense of surface coverage, and therefore, adhesion. However, analyzing single cells may provide insight on mechanistic behaviour of bacterial adhesion.

3.5 Cell Adhesion at the Single Cell Level

Whether or not cells are growing on the surface, it is evident that single cells or small colonies are continually adhering to the surface prefer attachment to other, already adhered cells. The number of small colonies dominate the number of large colonies, however, the large colonies contribute the largest, suggesting cells adhering onto the surface prefer attachment nearby or on colonies. This may be due to colonies that have adhered onto the surface has had time to significantly produce EPS or enabled quorum sensing to attract other cells [7, 8]. The EPS that are produced enable more availability for stronger, covalent attractions with incoming bacteria, allowing for preferential adhesion to larger colonies that are producing EPS [1]. As the method used in this study was not able to to continuous track of bacteria in real-time, imaging cannot directly determine if cells are growing once on the surface, however, coverage analysis suggests that during late exponential to stationary phase of growth, surface coverage of the largest colonies get significantly reduced, suggesting either cells are dispersing from the existing colonies or have stopped growth on the surface.

3.6 Cell Adhesion at the Microenvironment Level

A motivation for this imaging study was to complement previous nonlinear vibrational spectroscopy work that suggested that the interfacial water between water-silica, the bacterial microenvironment, are sensitive to microenvironment changes during this 24 h growth period [14]. The same spectroscopy experiment was done with only bulk media, which the bacteria were grown in, over time, to determine whether factors contributed by changing bulk media, had an impact. As seen in Figure 3.8(b), results showed that the bulk media (offline, no cells), minus bacteria, had no change in the interfacial water, indicating little contribution to the bacterial microenvironment. This suggests that bacteria must be adhering and growing to contribute to the change in interfacial water. The limitation of the spectroscopy study was that it could not directly visualize the bacterial surface over time, however, the surface coverage study solved this problem by designing a methodology complimentary to this previous work.

The spectroscopy work also showed that the trend in interfacial water signal increase match the same growth trend as bulk cell growth. As seen in Figure 3.9, comparing the bulk cell growth with surface coverage, surface coverage does not correlate with the interfacial water signal trend or bulk growth; surface coverage increases primarily during lag to mid-exponential phase, whereas interfacial water signal increases much slower, matching more inline with the growth curve. This suggests that bacteria alone, and the cell surface molecules that the bacteria bring along, cannot contribute to the change in water microenvironment. However, the spectroscopy study showed that bacteria must be present for the interfacial water signal to change. A possible model of adhesion could be that bacteria are first adhering onto the surface during the early phases of growth, and bacteria are producing adhesive products in later stages of growth that alter the bacterial microenvironment. Possibilities for these products include EPS and biofilms that bacteria are known to produce during cell adhesion. As colonies grow on the surface, bacteria continually uptake nutrients and produce EPS and byproducts that would be located in the

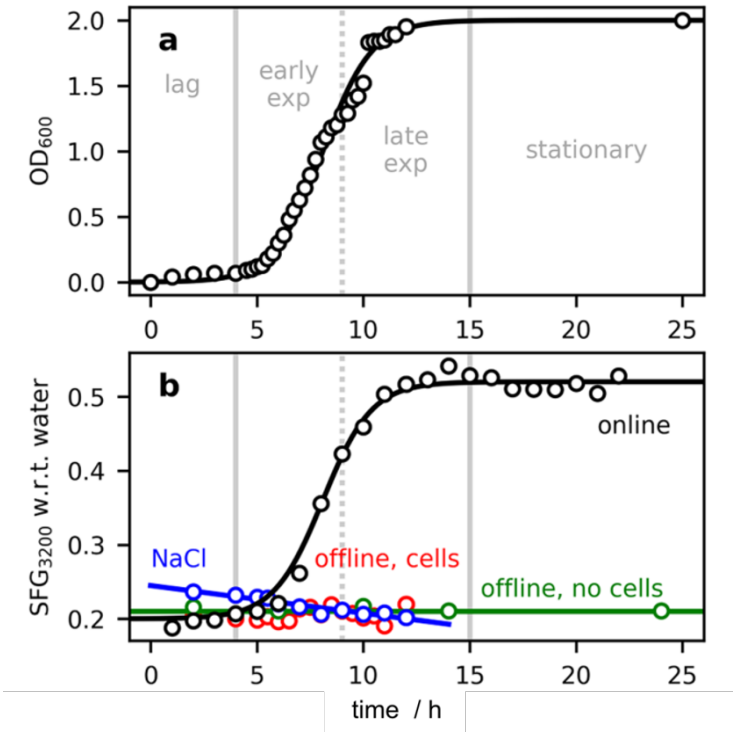

Figure 3.8: (a) OD600 bulk growth curve of *E. coli*. (b) Nonlinear vibrational response of the interfacial water O–H stretching signal at 3200 cm^{-1}. Adapted from Ref. 14.

bacteria-substrate microenvironment, causing changes in ionic strength and hydrophobicity that may have contributing factors that would match inline with previous spectroscopic and imaging results [1].

3.7 Adhesion Models

Previous studies suggest that bacteria and *E. coli* utilize biopolymers secreted on their cell surface, such as bound EPS, to help enhance adhesiveness onto surfaces. Other studies suggest that within the aqueous environment, surface conditioning occurs, where proteins, lipids, and ions first adhere to the surfaces allowing for stronger attraction of bacteria onto the surface, these molecules could be contributed by free EPS or other nutrients from media which bacteria are grown. Multiple models of bacterial adhesion are considered: cells with bound EPS adhere to the surface, free EPS condition the surface, or cells produce EPS after adhesion, as seen in Figure 1.1. Figure 3.10 showed that pH and ionic strength of

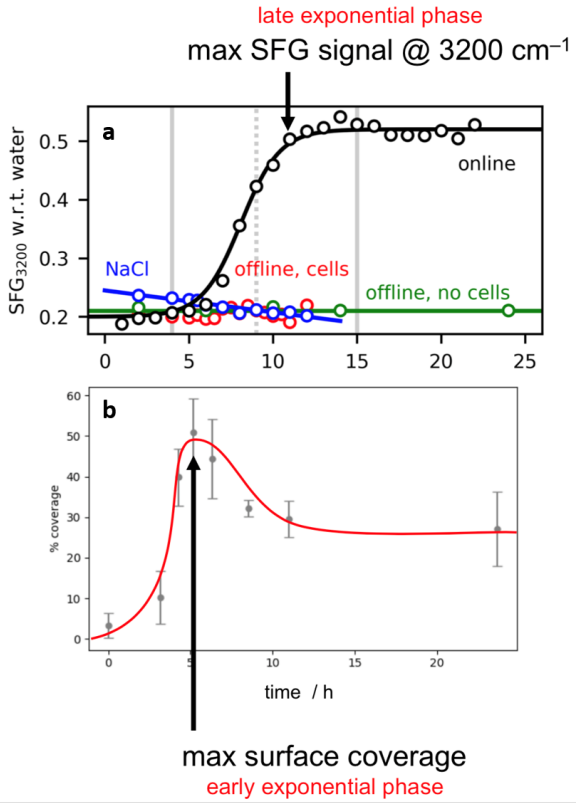

Figure 3.9: (a) Interfacial water signal over time, adapted from Ref. 14 (b) Percent coverage of bacteria over time.

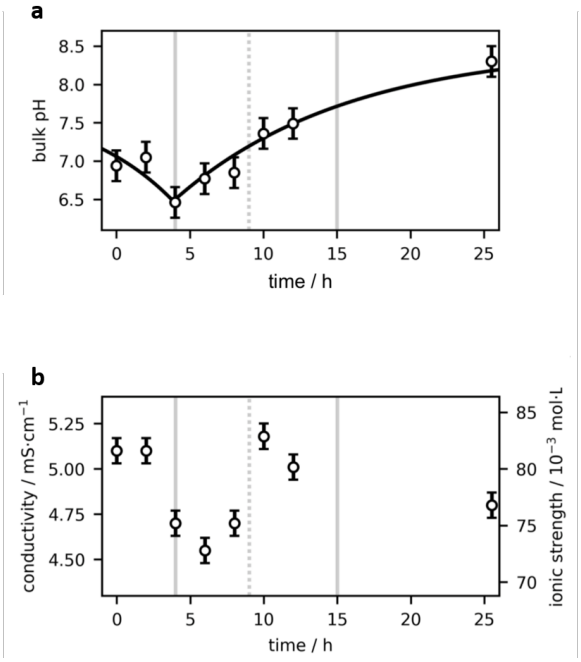

Figure 3.10: (a) Bulk media pH change over time. (b) Bulk media ionic strength change over time. Adapted from Ref. 14.

the growth media were tested, determining that these had little significance to the water signal contribution, and therefore may not be affecting the microenvironment. Other studies show that some products such as uronic acids, a component of EPS, increases in free EPS, as seen in Figure 3.11. However, this trend of uronic acid increases do not match the bacterial coverage, suggesting that, conditioning scenario may not play a primary role in adhesion [39]. Recent nonlinear vibrational spectroscopy and the imaging results suggest that bacterial adhesion may not be primarily enhanced by free EPS conditioning the surface or cells with bound EPS adhering to the surface. The lag between the maximum coverage of bacteria to water signal increases suggest that the microenvironment begins to change once the bacteria have some time to adhere to the surface, allowing for production of EPS, or changes in this microenvironment.

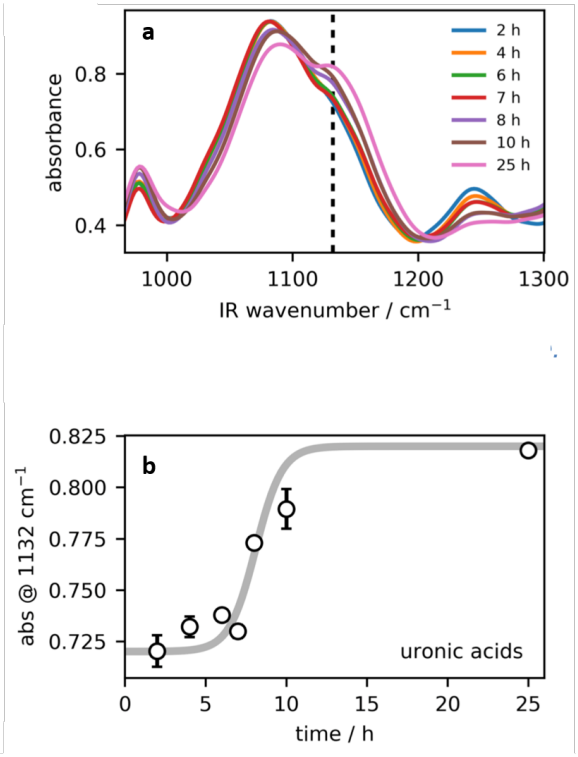

Figure 3.11: (a) FTIR spectra of free EPS from bulk growth solution over time. (b) FTIR response of uronic acid peak of 1132 cm^{-1} over time. Adapted from Ref. 39

Chapter 4

Raman Scattering of Adsorbed Bacteria

4.1 Background

To get a clearer understanding of the models previously proposed, imaging results are ideally complemented with chemical information that can distinguish whether specific molecules, such as those in EPS, play a role in the adhesion. Raman spectroscopy can be used to identify specific biomarkers in EPS from *E. coli* on silica, to determine whether EPS secreted by growing bacteria are adhered on the silica. The first objective was to extend the capabilities of the custom-built microscope to include Raman spectroscopy and imaging. The second objective was to utilize Raman spectroscopy to identify specific chemical biomarkers responsible for bacterial adhesion on silica surfaces, based on their bulk stages of growth.

The Raman effect is based on inelastic scattering between incident light when it interacts with molecules, where the scattered light of interest is at frequencies shifted from the incident frequency, referred to as Stokes and anti-Stokes Raman scattering. This spectroscopic technique is widely used for providing chemical composition and molecule identification [40]. For biological applications, Raman spectroscopy is ideal as a non-invasive, label-free technique that can probe the composition of *E. coli* and the EPS that they produce [41].

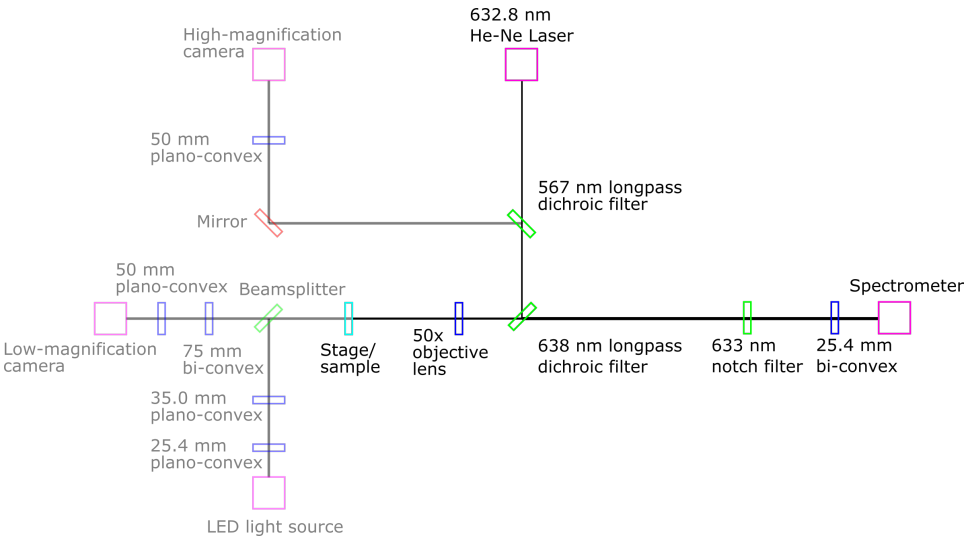

Figure 4.1: Schematic illustrating the main components of the confocal Raman light paths of the microscope.

4.2 Raman Microscope Development

The original design goal was to include Raman spectroscopy and imaging in the multimodal microscope described above. The intention was to enable future growth experiments in flow cells that track bacterial adhesion and colony formation over a period of weeks. Bacteria such as *E. coli* are known to produce little EPS compared to other species, and therefore take longer to establish biofilms [34]. As shown in Figure 4.1, a 632.8 nm He-Ne laser was focused using a 50× NA 0.55 long-working objective lens (Plan L, Nikon). A spectrometer consisting of a 750 mm monochromator (Acton SpectraPro 2750) was used in conjunction with a 1340 × 400 pixel CCD array (Princeton Instruments Pixis 400BR) to collect the Raman signal. The system provided good quality Raman spectra of bulk samples with sufficient thickness, but was unable to achieve sufficient signal-to-noise for sub-micron thick films of polystyrene or poly(dimethylsiloxane). This was likely due to further optimization of the collection efficiency that needed to be performed. As a result, I was not able to collect spectra from *E. coli*, and I have therefore used a commercial Raman system (Renishaw inVia) for the remainder of my experiments.

4.3 Experimental Procedure

E. coli were cultured with the same procedure as the previous imaging bacterial adhesion study. A change in procedure was replacing coverslips with one 1" $\times$ 2" $\times$ 0.5 mm crude-grade quartz microslide (Quartz Plus). This quartz microslide enabled higher signal-to-noise of Raman spectra due to the lower silica background fluorescence that this higher grade microslide had compared to disposable glass coverslips.

Bulk cell population was monitored using OD600. The quartz microslide was removed at specific time points where the bulk cell population were identified at being in lag, exponential, and stationary phases, which were at approximately at OD600 values of 0.1, 1.0, and 2.0 respectively. The microslide was dip rinsed in deionized water, and left to dry before Raman spectra acquisition.

An 500 mW 532 nm laser was used at 10% power, 60 s acquisition time, averaging 10 accumulations. Selecting the 2400 grooves/mm grating resulted in a spectral coverage of 551–589 nm, corresponding to a Stokes shift of 628–1808 cm^{-1}.

For Raman studies of different sized bacterial colonies, spectra were averaged with five different, but same number of bacteria clusters. For Raman spectra of bacteria at different stages of bulk growth, Raman spectra were averaged with five different, but same number of bacteria clusters.

For spatial rastering cross-sections of bacteria, the step size used was 0.5 μm. The spectra were colour coded based the brightfield image, to determine whether the laser spot was located on, near, or off the cell. Background correction was done using a method adapted from utilizing poly and contaminate fitting, and the spectra were smoothed using a Savitzky-Golay filter [42].

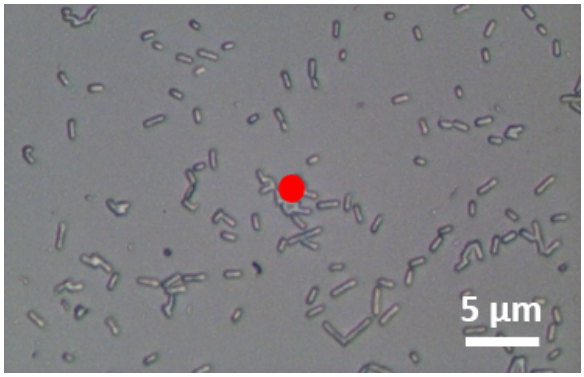

Figure 4.2: Brightfield image of *E.coli* on silica glass.

4.4 Results and Discussion

Figure 4.2 showed a brightfield image of *E. coli* on silica window with a red spot approximating where the incident laser of the microscope will be probing the surface. Raman spectra were collected manually from colonies of cells labelled as small, med, and large; representing clusters of bacteria that appeared to be one to two cell sizes, four to six cell sizes, and ten or more cell sizes respectively. The hypothesis was that since bacteria adhere individually over time, larger clusters of bacteria are more likely to have been adhered to the surface for longer, allowing time for EPS to be produced. If so, Raman spectroscopy may be able to detect changes in identifiable biomarkers among them. The results, as seen in Figure 4.3 show little difference between the differently sized clusters. There are visible peaks associated with phenylalanine at 1000 cm^{-1}, proteins at 1250 and 1350 cm^{-1}, lipids at 1450 cm^{-1}, and proteins at 1700 cm^{-1} [43]. However, these could also be associated with the chemical composition of the bacteria, not only the EPS that may be on the surface of the bacteria.

To investigate whether we could detect EPS changes on bacteria over time, Raman spectra were collected of bacteria at lag, exponential, and stationary phase. Figure 4.4 showed the Raman results, showing very little deviation among the three differing stages, and insufficient to determine whether the amount of EPS produced at these three different stages differ.

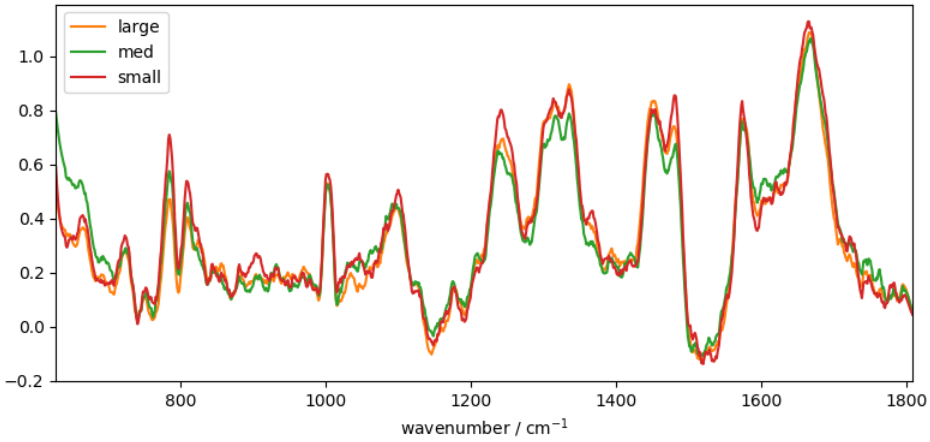

Figure 4.3: Raman spectra of small, med, large sized clusters of *E. coli*.

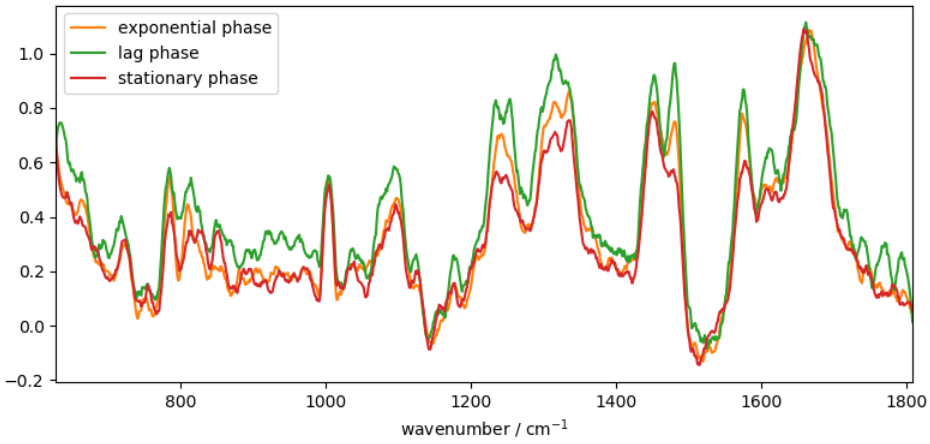

Figure 4.4: Raman spectra of *E. coli* during lag, exponential, and stationary phase of bulk growth.

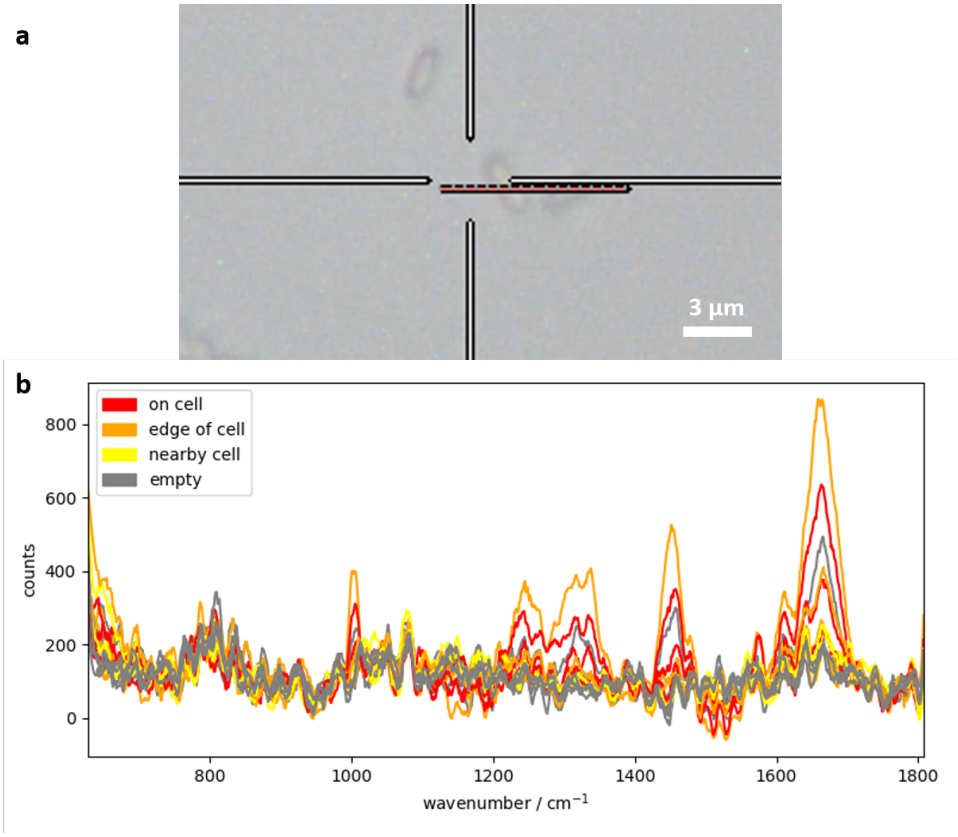

Figure 4.5: (a) Brightfield image with cross-section of two side-by-side bacteria to be probed spatially. (b) Raman spectra comparison at different spatial localization for bacteria.

It was hypothesized that if EPS was produced and secreted to the outside of bacterial colonies, probing the perimeter around *E. coli* may yield Raman spectra representative of EPS production. Figures 4.5 and 4.6 showed the results of this experiment. The microscope was rastered over the cross-section of the bacterium, and manually colour-coded to represent spectra that are identified as either probing the bacterium, the edge of the bacterium, nearby, and far away (empty).

The results of collecting spectra over a region that spanned two bacterial cells yielded no Raman signal for spectra localized beside, nearby, or far away from the bacterium. Spectra localized on the bacterium produced significant signal. Similar results were obtained by probing larger clusters containing 5–6 bacteria, as seen in Figure 4.6. This suggests that either EPS was not detectable using this method of Raman spectroscopy or that EPS was not being produced on the surface. However, this does not rule out the ability that bacteria are

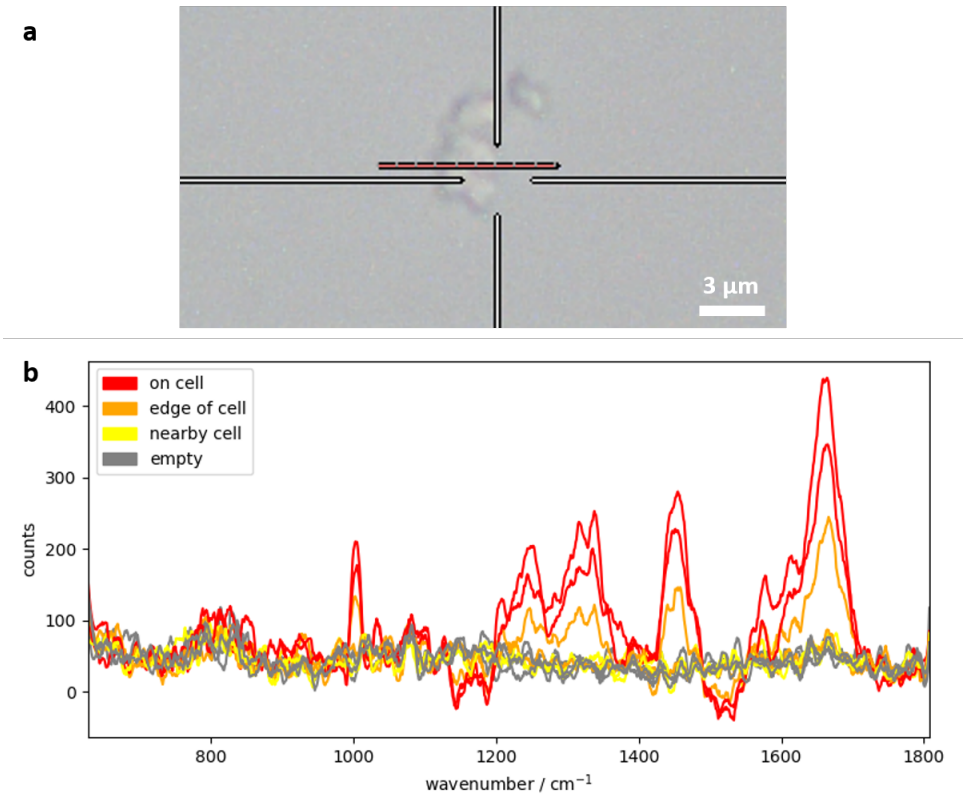

Figure 4.6: (a) Brightfield image with cross-section of multiple, clustered bacteria to be probed spatially. (b) Raman spectra comparison at different spatial localization for bacteria colony.

able to adhere on top of existing bacterial colonies. Since these surface-adhered bacteria are estimated to be microns, monolayers thick, with their EPS being even smaller, it is possible that this method cannot detect these changes on the surface and further improvement on this spectroscopic technique will be required.

Chapter 5

Conclusions

5.1 Summary of Work

Although bacterial adhesion and biofilm growth have been widely studied, more detailed cellular mechanisms and chemical environment contributions of bacterial adhesion on surfaces are still to be known. Previous work utilizing spectroscopy provided chemical information within the microenvironment of early bacterial adhesion on silica surfaces, however, no direct imaging technique was able to probe the specific bacteria-substrate surface.

This work complements prior nonlinear vibrational spectroscopy studies to directly image the bacteria and their adhesion onto silica surfaces over time. This work provided a custom imaging platform with complete software capabilities to perform high-throughput image acquisitions. Although initial development on the imaging system to include capabilities for confocal Raman microscopy was carried out, further optimization is required in order to generate spectra from thin samples.

The results of bacterial adhesion imaging suggest that surface coverage is mainly contributed by large colonies, whether they are incoming bacteria preferring adhesion to existing or nearby cells or due to cells dividing on the surface. As well, surface coverage was identified to increase during the earlier stages of growth, while previous work with nonlinear vibrational spectroscopy suggest that the interfacial water signal increases much more slowly, matching the bulk growth curve. This suggests that bacteria on the surface

are producing EPS that contributes to the attraction of bacteria onto existing colonies, and hence the change in interfacial water signal. This provided insight on the proposed model of adhesion where bacteria must first be adhering to the surface and given time, secretion of EPS enabled continued attraction of planktonic bacteria to adhere to the surface.

It was hypothesized that if *E. coli* are producing EPS, Raman spectroscopy is capable of identifying specific biomarkers on or within the bacteria that can contribute to bacterial adhesion. Using a conventional confocal Raman microscope to study bacteria on glass, results during lag, exponential, and stationary phases showed no significant difference in spectra. Spatial rastering of Raman spectra to identify whether EPS biomarkers can be identified outside of the bacteria or colony boundaries bacteria were carried out. No significant Raman spectra were observed outside of the cells, providing evidence for our observation that cells join existing colonies.

5.2 Future Work

Future directions of this work could aim to further improve the imaging platform by implementing and optimizing more complex probing techniques such as Raman microscopy, or optimizing the data acquisition software to achieve higher-throughput analysis of samples. The results presented here represent 2D projections of the adsorbed cells and surface-growing colonies. For bacterial adhesion studies, additional probes such as fluorescence could provide additional chemical evidence of bacterial adhesion mechanisms. For studies of bacterial biofilms, a more direct method of biofilm quantification can also be implemented such as a fluidic flow channel that would enable the continuous tracking of bacteria as they grow within the imaging platform. A more complete characterization could be performed using confocal fluorescence to track the growth of colonies in the direction perpendicular to the substrate surface. Further analysis of the Raman spectra can be achieved through statistical methods such as principal component analysis to extract specific wavenumber regions which indicate greatest variance and change in the spectra.

References

[1] Dunne, Jr., W. M. *Clin. Microbiol. Rev.* **2002,** *15,* 155–166.

[2] Berne, C.; Ellison, C. K.; Ducret, A.; Brun, Y. V. *Nat. Rev. Microbiol.* **2018,** *16,* 616–627.

[3] Katsikogianni, M.; Missirlis, Y. F. *Eur. Cell. Mat.* **2004,** *8,* 37–57.

[4] Li, J.; McLandsborough, L. A. *Int. J. Food Microbiol.* **1999,** *53,* 185–193.

[5] Nel, A. E.; Maedler, L.; Velegol, D.; Xia, T.; Hoek, E. M. V.; Somasundaran, P.; Klaessig, F.; Castranova, V.; Thompson, M. *Nat. Mater.* **2009,** *8,* 543–557.

[6] Donlan, R. M. *Emerg. Infect. Dis.* **2002,** *8,* 881–890.

[7] Davies, D. G.; Parsek, M. R.; Pearson, J. P.; Iglewski, B. H.; Costerton, J. W.; Greenberg, E. P. *Science* **1998,** *280,* 295–298.

[8] Barrios, A. F. G.; Zuo, R.; Hashimoto, Y.; Yang, L.; Bentley, W. E.; Wood, T. K. *J. Bacteriol.* **2006,** *188,* 305–316.

[9] Flemming, H.-C.; Wingender, J. *Nature Rev.* **2010,** *8,* 623–633.

[10] Shenga, G.-P.; Yua, H.-Q.; Li, X.-Y. *Biotechnol. Adv.* **2010,** *28,* 882–894.

[11] Eboigbodin, K. E.; Biggs, C. A. *Biomacromolecules* **2008,** *9,* 686–695.

[12] Davey, M. E.; O'toole, G. A. *Microbiol. Mol. Biol. Rev.* **2000,** *64,* 847–867.

[13] Azeredo, J.; Oliveira, R. *Biofouling* **2000,** *16,* 59–67.

[14] Jarisz, T. A.; Lane, S.; Gozdzialski, L.; Hore, D. K. *J. Chem. Phys.* **2018,** *148,* 222825.

[15] Jarisz, T.; Roy, S.; Hore, D. K. *Acc. Chem. Res.* **2018,** *51,* 2287–2295.

[16] Gali, S.; Garca-Gutirrez, C.; Migulez, E. M.; Villar, C. J.; Lomb, F. *Front. Microbiol.* **2018,** *9,* 1–18.

[17] Høiby, N.; Ciofu, O.; Johansen, H. K.; Song, Z.-j.; Moser, C.; Jensen, P. Ø.; Molin, S.; Givskov, M.; Tolker-Nielseni, T.; Bjarnsholt, T. *Int. J. Oral Sci.* **2011,** *3,* 55–65.

[18] Muffler, K.; Lakatos, M.; Schlegel, C.; Strieth, D.; Kuhne, S.; Ulber, R. *Adv. Biochem. Eng./Biotechnol.* **2014,** *146,* 123–161.

[19] Azeredo, J. *et al. Crit. Rev. Microbiol.* **2017,** *43,* 313–351.

[20] Binnig, G.; Quate, C. F.; Gerber, C. *Phys. Rev. Lett.* **1986,** *56,* 930–933.

[21] Bulard, E.; Fontaine-Aupart, M.-P.; Dubost, H.; Zheng, W.; Bellon-Fontaine, M.-N.; Herry, J.-M.; Bourguignon, B. *Langmuir* **2012,** *28,* 17001–17010.

[22] Uchida, S. *Dev. Growth Differ.* **2013,** *4,* 523–549.

[23] Coelho, L. P.; Shariff, A.; Murphy, R. F. *Proc. IEEE Int. Symp. Biomed. Imaging* **2009,** *5193098,* 518–521.

[24] Meijering, E. *IEEE Signal Process. Mag.* **2012,** *29,* 140–145.

[25] Caicedo, J. C. *et al. Nat. Methods* **2017,** *14,* 849–863.

[26] Bian, S. "Python interface to Thorlab's APT motion controllers", `https://github.com/freespace/pyAPT`, 2015.

[27] "Interface for IDS machine vision cameras", `https://github.com/ncsuarc/ids`, 2013.

[28] Sathyaa, P. D.; Kayalvizhib, R. *Eng. Appl. Artif. Intell.* **2011,** *24,* 595–615.

[29] Otsu, N. *IEEE Trans. Sys. Man Cybernetics* **1979,** *1*, 62–66.

[30] Cunliffe, D.; Smart, C. A.; Alexander, C.; Vulfson, E. N. *Appl. Environ. Microbiol.* **1999,** *65,* 4995–5002.

[31] Reshes, G.; Vanounouy, S.; Fishovz, I.; Feingold, M. *Biophys. J.* **2008,** *94,* 251–264.

[32] Kubitschek, H. E. *J. Bacteriol.* **1990,** *1*, 94–101.

[33] Sezonov, G.; Joseleau-Petit, D.; D'Ari, R. *J. Bacteriol.* **2007,** *189,* 8746–8749.

[34] Sung, B. H.; Lee, C. H.; Yu, B. J.; Lee, J. H.; Lee, J. Y.; Kim, M. S.; Blattner, F. R.; Kim, S. C. *Appl. Environ. Microbiol.* **2006,** *72,* 3336–3342.

[35] BetseyPitts,; Hamilton, M. A.; Zelver, N.; S.Stewart, P. *J. Microbiol. Methods* **2003,** *54,* 269–276.

[36] Marshall, W. F.; Young, K. D.; Swaffer, M.; Wood, E.; Nurse, P.; Kimura, A.; Frankel, J.; Wallingford, J.; Walbot, V.; Qu, X.; Roeder, A. H. K. *BMC Biol.* **2012,** *10,* 101–123.

[37] Grossman, N.; Ron, E. Z.; Woldringh, C. L. *J. Bacteriol.* **1982,** *152,* 35–41.

[38] Haznedaroglua, B.; Bolsterb, C. H.; Walker, S. L. *Water Res.* **2008,** *42,* 1547–1554.

[39] Hennecker, C. D.; Jarisz, T. A.; Hore, D. K. *J. Phys. Chem. C* **2019,** *123*, 6635–6641.

[40] Opilik, L.; Schmid, T.; Zenobi, R. *Annu. Rev. Anal. Chem.* **2013,** *6,* 379–398.

[41] Choo-Smith, L.-P. *et al. Appl. Environ. Microbiol.* **2001,** *67,* 1461–1469.

[42] Beier, B. D.; Berger, A. J. *Analyst* **2009,** *134,* 1198–1202.

[43] Huang, W. E.; Griffiths, R. I.; Thompson, I. P.; Bailey, M. J.; Whiteley, A. S. *Anal. Chem.* **2004,** *76,* 4452–4458.

Appendix A

Code Listings

A.1 Motor and Camera Control

Code A.1 controls the motors. Options to either import module or to directly use the script to input x, y, z positions or steps are enabled. Options to change velocity, home position are available.

Listing A.1: High-level motor control for MTS25.

```
# turn on all three motors
# for direct use: use control()
# raw input (two): 'motor step' -- moves by stepsize
# raw input (three): 'motor goto position' -- moves to
   position
# home motors to zero origin, cloes motors after use

import pyAPT, time, pickle

def _init_motors():
   # initilizes x, y, z motors
   x = pyAPT.MTS50(serial_number = 27001234)
   y = pyAPT.MTS50(serial_number = 27001328)
   z = pyAPT.MTS50(serial_number = 27001291)

   '''
   for each in motorlist:
      each.max_velocity = velocity = 0.23
      each.max_acceleration = 0.45
   '''
   print x.position(), y.position(), z.position()
   return x, y, z
```

```
def new_move(motor, step, delay=False):
    # move by step size
    # only used with control, adds delay
    motor.move(step)
    if delay == True:
        time_delay = abs(step)/0.23 + 2 # assume 0.23 mm/s,
            plus 2 s extra
        time.sleep(time_delay) # TURN OFF FOR FASTER RESPONSE
            BUT WATCH FOR CONFLICTS

def new_goto(motor, newposition, delay=False):
    # move by position
    # only used with control, adds delay
    motor.goto(newposition)
    if delay == True:
        time_delay = abs(newposition - motor.position())/0.23
            + 2
        time.sleep(time_delay)

def savemotors(filename='savexyz.pickle', delay=False):
    # save coords in pickle, and home motors to position 0
    # always run before turning off motors
    coords = [x.position(), y.position(), z.position()]
    print coords
    if raw_input('␣pickle␣coords?␣(y/n):␣') == 'y':
        pickle.dump(coords, open(filename, 'wb'))
        print('␣saved␣coords')
    else:
        print('␣not␣saved')
    homing = raw_input('␣home␣xyz?␣(y/n):␣')
    if homing == 'y':
        new_goto(z, 0.0000) # z first so no clashing
        for each in [x, y]:
            new_goto(each, 0.0000)
        if delay == True:
            longest = max([x.position(), y.position(), z.
                position()])
            time_delay = abs(longest)/0.23 + 2
            print time_delay
            time.sleep(time_delay)
        print('␣homed␣coords:␣%f,␣%f,␣%f' %(x.position(), y.
            position(), z.position()))
    else:
        print('␣not␣homed')
```

```
def loadmotors(filename='savexyz.pickle', delay=False):
    # print coords & homes motors back to previous position (
       from position 0, not rev. limit)
    # run after turning on motors to get back to position (if
        desired)
    coords = pickle.load(open(filename, 'rb'))
    print('␣pickled␣coords:␣%f,␣%f,␣%f' %(coords[0], coords
       [1], coords[2]))
    if raw_input('␣rev.␣home␣xyz?␣(y/n):␣') == 'y':
       i = 0
       for each in [x, y, z]:
          new_goto(each, coords[i])
          i = i + 1
       if delay == True:
          time_delay = abs(max(coords))/0.23 + 2 # assume
             0.23 mm/s with 2 s
          time.sleep(time_delay)
       print('␣rev.␣homed:␣%f,␣%f,␣%f' %(x.position(), y.
          position(), z.position()))
    else:
       print('␣not␣rev.␣homed')

def control():
    # for continuous control inputting args: 'motor step' for
        step movement, 'motor goto position' for specific
       position
    # default input: z down 0.01 mm
    motor = z
    step = .0100

    try:
       while True:
          inputs = raw_input('␣input␣move␣(motor,␣step):␣')
          inputs = inputs.split()
          if len(inputs) >= 2:
             if inputs[0] == 'x':
                motor = x
             elif inputs[0] == 'y':
                motor = y
             elif inputs[0] == 'z':
                motor = z
             else:
                print '␣motor␣error'

             if len(inputs) == 3: # three inputs: go to
```

```
                position
            #if inputs[1] == 'goto':
                newposition = float(inputs[2])
                new_goto(motor, newposition)
                print ('␣%s␣position:␣%f␣mm' %(inputs[0],
                    motor.position()))
            elif len(inputs) == 2: # two inputs: move steps
                step = float(inputs[1])
                new_move(motor, step)
                print ('␣%s␣position:␣%f␣mm' %(inputs[0],
                    motor.position()))
         if len(inputs) == 0: # no input: repeat default/
            last new_move input
            new_move(motor, step)
            print ('␣r␣position:␣%f␣mm' %(motor.position()))
   except KeyboardInterrupt:
      print '\n␣control␣done'

def closemotors():
   # is this really needed?
   for each in [x, y, z]:
      each.close()
      print '␣motors␣closed'

### MAIN CODE ###
if __name__ == '__main__':
   # run directly for options to load, save, or control
   x, y, z = _init_motors()
   if raw_input('␣load?␣(y/n):␣') == 'y':
      loadmotors(delay=True)
      control()
   else:
      control()
   time.sleep(5)
   if raw_input('␣save?␣(y/n):␣') == 'y':
      savemotors(delay=True)
   closemotors()
   print('motorcontrol.py␣ran␣directly')
else:
   x, y, z = _init_motors()
   print('motorcontrol.py␣imported.␣x,␣y,␣z')
```

Code A.2 is an example of running the code A.1 directly from the Linux terminal, allowing inputs to control motors and stages, and then homing the position back to the

origin after use.

Listing A.2: Example of controlling stages and motors.

```
dkhlab@dkhlab-scope3:/mnt/cluster-victor/lin_motors$ python
   motorcontrol.py
0.0 0.0 0.0
 load? (y/n): y
 pickled coords: 1.000000, 1.000000, 3.000000
 rev. home xyz? (y/n): y
 rev. homed: 1.000000, 1.000000, 3.000029
 input move (motor, step): z 1.00
 z position: 3.000000 mm
 input move (motor, step): z goto 1.00
 z position: 4.000000 mm
 input move (motor, step): ^C
 control done
 save? (y/n): y
[1.0, 1.0, 1.0]
 pickle coords? (y/n): n
 not saved
 home xyz? (y/n): y
6.34782608696
 homed coords: -0.001691, 0.000146, -0.021572
 motors closed
```

Code A.3 controls IDS cameras. It can be imported as a module or ran directly to turn on video preview. Option to change exposure time is enabled.

Listing A.3: High-level camera control for IDS camera.

```
### ids camera functions
# always run start_cam() first
# camPreview to do live previewing, 1s per frame
# close fig by ctrl+c
# close cam before turn off

import ids, pylab, time, cv2, sys, datetime

def start_cam(ida, expo=0, pixelclock=24):
   # turn on cam, note default exposure time and pixelclock
   cam = ids.Camera(ida)
   cam.color_mode = ids.ids_core.COLOR_RGB8 # preview blue,
      saved yellow
   #cam.color_mode = ids.ids_core.COLOR_BGR8 # preview
      yellow, saved blue
```

```
    cam.pixelclock = pixelclock # reduce pxclock to increase
        expo
    if expo > 0:
        cam.exposure = expo
    elif expo == 0:
        cam.exposure = 50
        #cam.auto_exposure = True
    return cam

def take_pic2(cam, autosave=False, direct=''):
    # takes a picture after x delay, throwing out first five
        frames due to buffer
    expo = cam.exposure
    time.sleep(expo/1000 * 3) # delay to ensure correct
        exposure
    cam.continuous_capture = True
    for i in range(5): # have to run through buffered frames
        img, meta = cam.next()
    img, meta = cam.next()
    if autosave == True:
        filename = '%s%s.jpg' %(direct, datetime.datetime.
            strftime(datetime.datetime.now(), '%Y%m%d_%H%M%S'))
        cam.next_save(filename)
    cam.continuous_capture = False
    return img, expo

def display_img(fig, img):
    # requires fig = pylab.figure()/pylab.ion() for
        continuous viewing
    pylab.clf()
    ax1 = fig.add_subplot(111)
    ax1.imshow(img)
    ax1.figure.canvas.draw()

def save_img(filename, img):
    # save bgr or rgb?
    img2 = img
    img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
    cv2.imwrite(filename, img2)

def camPreview(direct='/mnt/cluster-victor/lin_motors/pics/
    raster/'):
    # for pylab previewing/ueye alternative: option to save
        last frame
    # note the default save directory
```

```
    check = raw_input('␣preview␣starting...␣close␣ueye.␣
       continue?␣(y/n)␣')
    if check == 'y':
       pylab.ion()
       fig = pylab.figure()
       # run cam
       cam = start_cam(2)
       conts = 0
       try:
          while True:
             img, expo = take_pic2(cam, False, direct)
             display_img(fig, img)
             fn = '/mnt/cluster-victor/lin_motors/pics/video/
                d%s_c%003d.png' %(datetime.datetime.now().
                strftime('%Y%m%d_%H%M%S'), conts)
             conts = conts + 1
             save_img(fn, img)
       except KeyboardInterrupt:
             display_img(fig, img) # display last frame
       cam.close()
       # saving file
       filename = '%stest_exp_%.2f.png' %(direct, expo)
       if raw_input('␣save␣img:␣%s?␣(y/n):␣' %filename) == 'y
          ':
          save_img(filename, img)
       else:
          print('␣no␣save')


### MAIN CODE
#direct = '/mnt/cluster-victor/lin_motors/pics/raster/'
if __name__ == '__main__':
    # use directly for video preview
    if raw_input('␣turn␣on␣camPreview?␣(y/n):␣') == 'y':
       camPreview()
    else:
       print('camcontrol.py␣ran␣directly')
else:
    print('camcontrol.py␣imported.␣run␣start_cam(id)␣before␣
       using␣cam.')
```

Code A.4 is an example of running the script, code A.3 directly from the Linux terminal, turning on the video preview. Option to save the last frame of the video was shown.

Listing A.4: Example of controlling camera.

```
dkhlab@dkhlab-scope3:/mnt/cluster-victor/lin_motors$ python
   camcontrol.py
 turn on camPreview? (y/n): y
 preview starting... close ueye. continue? (y/n) y
^C save img: /mnt/cluster-victor/lin_motors/pics/raster/
   test_exp_50.00.png? (y/n): n
 no save
```

A.2 Autofocus and Raster Pattern Control

Code A.5 can be ran directly to capture 41 images in x stepsize of 1.0 mm, y stepsize of 2.0 mm, covering an area of 10 x 8 mm. The autofocus algorithm is encased in a class, with it's function run_autofocus2() to autofocus an image. The rastering pattern is encased in the function, zigzag(), requiring only the motor stepsize and the number of stepsizes to run.

Listing A.5: Autofocus and raster pattern.

```
# patterns and autofocus

# motorcontrol uses init x,y,z motors with new_move function
# camcontrol uses take_pic, save_img functions
# don't run from edge of motor, due to backcorrection

'''
filenaming() - in
combine() - in (no longer needed due to need for autofocus,
   use to compare no focus vs focus)
class autofocus() - in
   pic_sd() - in
   auto_pattern2() - in
   run_autofocus2() - in
   movefocus() - in
backnforth() - in, requires autofocus
zigzag() - in, requires autofocus
'''

import time, datetime
from camcontrol import *
from motorcontrol import *
```

```
import numpy as np

def filenaming(direct, motor, step, counter=0, printname=
   False):
   # create filename
   filename = '%sd%s_m%s_s%f_p%f_c%02d.png' %(direct,
         datetime.datetime.now().strftime('%Y%m%d-%H%M%S'),
         motor.serial_number[-5:-1],
         step, motor.position(), counter)
   if printname == True:
      print filename
   return filename

def combine(motor, step, counter, direct='', imaging=True):
   # move motor and take picture, requires counter
   # defunc: used more for testing and debugging
   if counter == 0 and imaging == True:
      # save first picture before movement
      img, expo = take_pic2(cam, False, direct)
      save_img(filenaming(direct, motor, step, counter), img
         )
   new_move(motor, step, True)
   time.sleep(1) # extra pause
   counter = counter + 1 # counter is global
   if imaging == True:
      img, expo = take_pic2(cam, False, direct)
      save_img(filenaming(self.direct, self.motor, self.step
         , printname=True), img)
   return counter

class autofocus:
   # starting sample at some point above focus
   # record three per plane, if best one is the middle, go
      to it and go to next range
   # if best is one of the sides, go to that one and repeat
      the same step
   # best to manual coarse focus first just for convenience
   # final img should be -0.02 mm offset
   # set step sizes in zranges

   # requires motor/cam connect, directory (where to save
      files), z ranges, autosave on/off
   def __init__(self, motor, cam, direct, zranges=[0.016,
      0.008, 0.004, 0.002, 0.001], autosave=False):
      self.motor = motor
```

```
        self.step = 0
        self.cam = cam
        self.direct = direct
        self.zranges = zranges
        self.autosave = autosave

    def pic_sd(self, imgs, list_sd, list_posi, rev=False):
        # calculates standard deviation of three images
        # moves motor +step, takes pic, appends img/sd to list
        # if img is saved, also print the filename
        if rev == False:
            new_move(self.motor, self.step, True)
        elif rev == True:
            new_move(self.motor, self.step * -1, True)
        elif rev == 'True2':
            new_move(self.motor, self.step * -2, True)
        #time.sleep(1)
        img, expo = take_pic2(self.cam, False, self.direct)
        global counter
        if self.autosave == True: # save every pic
            save_img(filenaming(self.direct, self.motor, self.
                step, counter, printname=False), img)
        #print(filenaming(self.direct, self.motor, self.step))
        img_slice = img[:,:,1] # select green slice
        posi = self.motor.position()
        #sd_norm = cvr.sd_img(img_slice)[0] # calculate sd (
            for debugging)
        sd_norm = np.std(img_slice.flatten()) # curde way
            without importing coverageRun
        # append img, sd, position to current plane's list
        imgs.append(img)
        list_sd.append(sd_norm)
        list_posi.append(posi)
        if self.autosave == True:
            print(list_posi)
            print(list_sd)
        #global counter
        counter = counter + 1
        return imgs, list_sd, list_posi

    def auto_pattern2(self, prev):
        # moves motors and captures images
        # pattern2: final img from previous iteration is used
            (increased speed)
        edge = True
```

```
        global counter
        while edge == True:
            imgs = []
            list_sd = []
            list_posi = []
            # one step down
            imgs, list_sd, list_posi = self.pic_sd(imgs,
               list_sd, list_posi)
            if prev == []:
               # if list is empty, must be first one: two steps
                   up
               #print('PREV IS EMPTY')
               for i in range(2):
                  imgs, list_sd, list_posi = self.pic_sd(imgs,
                     list_sd, list_posi, rev=True)
            else:
               # save previous iteration to middle of list
               imgs.append(prev[0])
               list_sd.append(prev[1])
               list_posi.append(prev[2])
               save_img(filenaming(self.direct, self.motor,
                  prev[2], counter, printname=False), prev[0])
                  if self.autosave == True else False
               counter = counter + 1 if self.autosave == True
                  else False
               #if self.autosave == True:
                  #print list_posi # for debugging
                  #print list_sd
               # one double-step up
               imgs, list_sd, list_posi = self.pic_sd(imgs,
                  list_sd, list_posi, rev='True2')
            sd_index = list_sd.index(max(list_sd)) # find max
               sd among three planes
            print sd_index
            if sd_index == 1: # if best pic is middle, end and
               return best slice
               new_goto(self.motor, list_posi[sd_index], True)
               #time.sleep(1)
               edge = False
               prev = imgs[1], list_sd[1], list_posi[1]
            elif sd_index == 2 or sd_index == 0:
               new_goto(self.motor, list_posi[sd_index], True)
               prev = imgs[sd_index], list_sd[sd_index],
                  list_posi[sd_index]
               #time.sleep(1)
```

```
            else:
                print('sd_index error, check code')
        return prev

    def run_autofocus2(self):
        # function for autofocus pattern and calculates best
            image
        print(' starting autofocus...')
        prev = []
        for zstep in self.zranges:
            self.step = zstep
            #print(self.step)
            prev = self.auto_pattern2(prev)
            if zstep == self.zranges[-1]:
                save_img('%sd%s_p%f_final.png' %(self.direct,
                    datetime.datetime.now().strftime('%Y%m%d-%H%M
                    %S'), self.motor.position()), prev[0])

    def movefocus(self, motor, step):
        # high-level function that autofocuses after moving to
            a new position
        # step and focus: autosave should be False, don't need
            to print list_posi/ each time,
        new_move(motor, step, True)
        #time.sleep(1)
        self.run_autofocus2() # should save final img


def backnforth(auto, motor, step, rep):
    # pattern rastering: back and forth, used for debugging
        and confirming precision of motor step sizes
    # will doubleback on all imgs but the middle img
    auto.run_autofocus2() # focus initial spot
    for i in range(rep):
        auto.movefocus(motor, step)
    for i in range(rep):
        auto.movefocus(motor, step * -1)
    # backnforth(auto, x, 0.5, 3) # example

def zigzag(auto, zig, zag):
    # main rastering pattern
    # zig/zag requires three inputs: [0] = motor, [1] = step,
        [2] = rep
    # long_rep = length of the actual zig, short_rep is the #
        of cycles/zag
```

```
    # pattern: zig x N, zag x 1, zig x -N, zag x 1
    # ---- zig
    # | zag
    # ---- zig
    # | zag
    # example:
    # zig = [y, 0.5, 3] -1 of zig since zag and initial
       run_autofocus2
    # zag = [x, 0.5, 2] one zag includes two rows of zig
    # zigzag(auto, zig, zag)
    auto.run_autofocus2()
    print(x.position(), y.position())
    for i in range(zag[2]):
        for i in range(zig[2]):
            auto.movefocus(zig[0], zig[1])
            print(x.position(), y.position())
        auto.movefocus(zag[0], zag[1])
        print(x.position(), y.position())
        for i in range(zig[2]):
            auto.movefocus(zig[0], zig[1] * -1)
            print(x.position(), y.position())
        auto.movefocus(zag[0], zag[1])
        print(x.position(), y.position())

def zigzag_range(zig, zag):
    # used just to determine whole range of x,y (area of
       zigzag rastering pattern)
    zig_range = zig[2] * zig[1] + zig[0].position()
    zag_range = zag[2] * zag[1] * 2 + zag[0].position()
    print(zig_range)
    print(zag_range)


### MAIN CODE ###
direct = '/mnt/cluster-victor/lin_motors/pics/raster
    /20190423/' # ENSURE CORRECT PATH TO SAVE IMGS

# set zigzag pattern parameters
imaging = True
if imaging == True:
    check = raw_input('␣raster␣starting...␣close␣ueye.␣\n␣(y/
       n):␣')
    if check == 'y':
        cam = start_cam(2)
        img, expo = take_pic2(cam, False, direct)
```

```
      # open autofocus class
      #autosave = True # to save every img
      autosave = False
      zranges = [0.016, 0.008, 0.004, 0.002, 0.001] # 0.016
      #zranges = [0.016, 0.008, 0.004, 0.002, 0.001]
      auto = autofocus(z, cam, direct, zranges, autosave)
   elif check != 'y':
      imaging = False

# run the main zigzag raster pattern
time.sleep(1)
global counter
counter = 0

if True:
   ''' note: y can't go past position 11 or will crash into
      cage '''
   zig = [y, 2.0, 4] # y-stepsize of 2.0 mm, moving four
      times (capturing five images)
   zag = [x, 1.0, 4] # x-stepsize of 1.0 mm, moving eight
      times... total of 5*8+1 = 41
   zigzag_range(zig, zag)
   zigzag(auto, zig, zag)
   #auto.run_autofocus2() # debugging: to confirm autofocus
      works correctly

# after pattern is complete, move back to initial position
time.sleep(10)
new_goto(x, 1)
new_goto(y, 1)
new_goto(z, 3)
time.sleep(60)

closemotors()
```

A.3 Image Thresholding Control

Code A.6 is imported to analyze images by image thresholding using a modified Otsu's method. The main function is cellcoverage_img() to threshold an image to result in the calculated percent coverage, number of colonies, and sizes of colonies. cellcoverage_multiavg() is to threshold multiple images, saving percent coverage, number of

colonies, and sizes of colonies in a list. Options to crop images, plot images, and plot pixel intensity histograms are available.

Listing A.6: Low-level scripts required for image thresholding.

```
# functions for cell coverage calculations

# -*- coding: utf-8 -*-
"""
lower the px value, the darker the colour
make white the bg, black the interest: plt white = 1, black
   = 0; interest = low px, bg = high px
"""

'''
load_img() - in
ax_labeloff() - in
show_imgs() - in
find_thresh() - in
sd_img() - in
entro_img() - in
otsu() - in
threshold_img() - in
threshold_img_cv2() - in
histo_img2() - in
crop_img() - in
y2b_save() - in
ezplot() - in
'''
import numpy as np
from matplotlib import pyplot as plt
import scipy, cv2, os
from scipy.ndimage import label

def load_img(filename, rgb=1, norm=False):
   # load img with cv2. if the geeqie picture is yellow: 0=r
      , 1=g, 2=b. if blue: 0=b, 1=g, 2=r
   # if geeqie colour is yellow, pylab colour is blue
   img = cv2.imread(filename)
   img = img[:,:,0:3] # get rid of alpha
   img_select = img[:,:,rgb]
   if norm == True:
      img_select = img_select / float(img_select.max()) #
         won't work with cv2 thresh
   return img, img_select
```

```
def show_imgs(bundle_imgs, titles=['']):
    # show images in tiles
    # always with two rows, requires imgs to be in a list, if
        run on linux, needs plt.show() at end of script, not
       function
    fig = plt.figure()
    count_imgs = len(bundle_imgs)
    if len(titles) == 1:
       titles = titles * count_imgs
    if count_imgs == 1:
       ncols = 1
       nrows = 1
    elif count_imgs > 1:
       ncols = int(round(count_imgs / 2.0))
       nrows = 2
    for ea in range(len(bundle_imgs)):
       ax = fig.add_subplot(nrows, ncols, ea+1)
       #ax.imshow(bundle_imgs[ea]) # original colour
       ax.imshow(bundle_imgs[ea], cmap='gray')
       ax.set_title(titles[ea])
       ax = ax.tick_params(
           labelleft=False,
           labelbottom=False,
           left=False,
           bottom=False)
    plt.tight_layout(True)
    #plt.show()
    return

def find_thresh(img, bins=0, preview=True, scale=True,
    img_orig=''):
    # histogram the distribution of px intensity of the img,
       to roughly guess threshold
    img_flat = img.flatten() # collect histogram
    if bins == 0:
       bins = img_flat.max() - img_flat.min()
    bin_counts, bin_edges = np.histogram(img_flat, bins)
    if preview == True:
       plt.rcParams['font.size'] = 6
       fig = plt.figure() # create plot
       ax1 = fig.add_subplot(222)
       ax1.imshow(img)
       ax1.set_title('img')
       ax2 = fig.add_subplot(212)
```

```
      if type(img_orig) == type(np.array([])):
         ax3 = fig.add_subplot(221)
         ax3.imshow(img_orig)
         ax3.set_title('original')
         #bin_counts_orig, bin_edges_orig = np.histogram(
            img_orig.flatten(), bins)
         #ax2.bar(bin_edges_orig[:-1], bin_counts_orig,
            color='orange')
      ax2.bar(bin_edges[:-1], bin_counts)
      ax2.plot(bin_edges[:-1], bin_counts)
      ax2.set_xlabel('px␣intensity')
      ax2.set_ylabel('counts')
      if scale == True:
         ax2.set_yscale('log')
      #ax2.hist(img_flat, bins) # same, but only good for
         plotting
      #ax2.set_ylim([0, 100])
      plt.tight_layout(True)
   return bin_counts, bin_edges

def sd_img(img, norm=True):
   img_flat = img.flatten()
   avg = np.mean(img_flat)
   sd = np.std(img_flat)
   if norm == True:
      sd = sd / float(avg)
   return sd, avg

def entro_img(bin_counts):
   # fake entropy, just favours img with largest single bin
   bin_counts_norm = bin_counts / float(max(bin_counts))
   entro_fake = sum(bin_counts_norm * bin_counts_norm)
   return entro_fake

def otsu(img, limit=0.50, limit_upper=0.70): # 0.90
   # otsu's can be used to roughly find a good threshold
      value
   # modified otsu: set upper limit where modified otsu does
       not affect
   ret, img_thresh = cv2.threshold(img, 0, 255, cv2.
      THRESH_BINARY+cv2.THRESH_OTSU) # if distribution is >=
       50%, then just do otsu

   # modify otsu
   bin_counts, bin_edges = np.histogram(img.flatten(), (img.
```

```
      flatten().max() - img.flatten().min()))
   mid = (img.flatten().max() - img.flatten().min())/2 + img
      .flatten().min() # split image px intensities by
      middle px intensity
   threshold_index = np.where(bin_edges == mid)[0][0]
   wf = np.sum(bin_counts[:threshold_index])
   wb = np.sum(bin_counts[threshold_index:])

   wf_percent = float(wf) / (wf + wb) # how much of lowest
      50% vs. total
   print(wf_percent)
   if wf_percent < limit:
      diff = ret - img.flatten().min()
      lim_percent = 1 - wf_percent # % of distribution
         relative to total
      multiplier = lim_percent ** 10 # multiply lim_percent
         by ^4 [lim_percent can never by higher than .50]
      ret = round(ret - multiplier * diff)

   elif wf_percent > limit_upper:
      diff = img.flatten().max() - ret
      lim_percent = wf_percent # % of distribution relative
         to total
      multiplier = lim_percent ** 10 # multiply lim_percent
         by ^4 [lim_percent can never by higher than .50]
      print(str(ret) + ' before ' + str(diff)+ ' ' + str(
         multiplier))
      ret = round(ret + multiplier * diff)
      print(str(ret) + ' after')
# print('%d otsu threshold' %ret)
   return ret, img_thresh

def threshold_img(img, threshold, norm=False):
   # thresholding, option to normal data first: returns b/w
      img to plot and data to histogram, and fraction value
   if norm == True:
      img_norm = img/float(img.max())
   else:
      img_norm = img
   img_mask = img_norm < threshold # anything lower than
      threshold, is of interest. as threshold decreases,
      interest gets tigheter
   #img_mask = np.invert(img_mask)
   #img_mask = img_mask.astype(int)
   #img_threshold = img_norm * img_mask
```

```
    img_threshold = np.invert(img_mask) # since interest is
        1, plt wants 1 to be white, need to inversed to be
        plotted nicely
    img_labelled, num_features = label(img_mask) # count 1's
    frac = (img_mask).sum() / float(len(img)*len(img[0])) *
        100
# print('%0.2f%% coverage | %d spots' %(frac, num_features))
    return img_threshold, img_labelled, frac

def threshold_img_cv2(img_slice, threshold, maxvalue=255):
    # only tried if input img is just one rgb slice, would a
        whole img work?
    ret, img_thresh = cv2.threshold(img_slice, threshold,
        maxvalue, cv2.THRESH_BINARY) #img is 0's and 255's
    img_prelabel = img_thresh < (maxvalue - 50) # convert 0's
         and 255's to T/F... -50 as a buffer for some reason
    img_labelled, num_features = label(img_prelabel)
    frac = (100 * (1-(img_thresh/maxvalue).sum() / float(len(
        img_thresh) * len(img_thresh[0])))) # 100 * # of 255's
         / tot # of px
# print('%0.2f%% coverage | %d spots - cv2' %(frac,
    num_features))
    return img_thresh, img_labelled, frac

def histo_img2(img, pxconverter=0.02562347656, bins=0,
    preview=True, crude=False, title='', img_rgb=''):
    # histogram the size of pixels with a labelled img b/w
    img_flat = img.flatten()
    #pxconverter = 0.160109
    #pxconverter = 0.02562347656 # 50x real
    #pxconverter = 0.00480651855 # 100x real
    counts_list = []
    if crude == True:
        for i in range(1, img_flat.max()+1):
            counts = list(img_flat).count(i) * pxconverter
            print counts
            counts_list.append(counts)
    elif crude == False:
        n_counts, counts_list = np.unique(img, return_counts=
            True)
        counts_list = counts_list[1:] * pxconverter # list of
            all labelled spots and their size: max should be 34
            k um^2

    if bins == 0:
```

```
      print n_counts
      bins = n_counts / 10
   if preview == True:
      plt.rcParams['font.size'] = 7
      fig = plt.figure()
      if type(img_rgb) == type(np.array([])):
         ax3 = fig.add_subplot(221)
         ax3.imshow(img_rgb)
         ax3.set_title('img')
      ax1 = fig.add_subplot(222)
      img2 = np.invert((img>1).astype(int))
      ax1.imshow(img2, cmap='gray') # ignore
      ax1.set_title('threshold')
      ax2 = fig.add_subplot(212)
      ax2.hist(counts_list, bins)
      ax2.set_yscale('log')
      ax2.set_xlim([0, max(counts_list)])
      ax2.set_xlabel('colony␣size␣/␣$\mu$m$^2$')
      ax2.set_ylabel('counts')
      ax2.set_title(title)
      plt.tight_layout(True)
   return counts_list

def crop_img(img, edge):
   # crop rgb image from center
   # for analyzing img
   h = len(img)
   w = len(img[0])
   img = img[edge:h-edge, edge:w-edge, :]
   return img

def y2b_save(filename, img):
   # convert yellow BGR to blue RGB image
   img2 = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
   cv2.imwrite(filename, img2)
   print('%s␣saved.' %filename)

def ezplot(img, title=''):
   # for analyzing img
   fig = plt.figure()
   ax = fig.add_subplot(111)
   ax.imshow(img, cmap='gray')
   ax.set_title(title)

def cellcoverage_img(filename, pxconverter, bins=50, preview
```

```
   =True):
   # cell coverage of one image, outputting colony size
      distribution, and fraction
   img, img_select = load_img(filename, 1)
   ret, img_otsu = otsu(img_select)
   threshold = ret # offset otsu value by 7
   img_thresh, img_labelled, frac = threshold_img_cv2(
      img_select, threshold)
   #pxconverter = 0.00480651855 # 100x
   #pxconverter = 0.02562347656 # 50x
   counts_list = histo_img2(img_labelled, pxconverter, bins,
       title='%0.2f%% coverage | %d threshold' %(frac,
      threshold), img_rgb=img_select, preview=preview)
   return counts_list, bins, frac

def cellcoverage_multiavg(filenames, pxconverter, bins=50,
   preview=True):
   # cell coverage of multiple images by outputting an
      average % coverage from distribution of % coverage
      from each image
   frac_list = []
   counts_list2 = []
   for f in filenames:
      img, img_select = load_img(f, 1)
      ret, img_otsu = otsu(img_select)
      threshold = ret
      img_thresh, img_labelled, frac = threshold_img_cv2(
         img_select, threshold)
      frac_list.append(frac)
      counts_list = histo_img2(img_labelled, pxconverter,
         bins, title='%0.2f%% coverage | %d threshold' %(
         frac, threshold), img_rgb=img_select, preview=
         preview)
      counts_list2.extend(counts_list)
   return frac_list, counts_list2

def cellcoverage_saveimgs(filenames):
   # cell coverage of multiple images by outputting an
      average % coverage from distribution of % coverage
      from each image
   # includes saving each thresholded img
   # for analyzing img
   img_list = []
   for f in filenames:
      img, img_select = load_img(f, 1)
```

```
      ret, img_otsu = otsu(img_select)
      threshold = ret
      img_thresh, img_labelled, frac = threshold_img_cv2(
         img_select, threshold)
      img_list.append(img_thresh)
   return img_list


def save_bwimg(filename):
   # save a bw img of an example img
   #filename = '/mnt/cluster-victor/lin_motors/pics/.....png
      '
   img, img_select = load_img(filename, 1)
   ret, img_otsu = otsu(img_select)
   threshold = ret - 7
   img_thresh, img_labelled, frac = threshold_img_cv2(
      img_select, threshold) # this is ran twice
   cv2.imwrite('/mnt/cluster-victor/analysis/counting/
      ecoli_test.png', img_thresh)
   print( 'bw␣img␣saved.')
   cellcoverage_img(filename, 0.256235, 100, True)
   plt.show()

def plot_twincurve(duration_od600, od600, duration_coverage,
    coverage, fig=''):
   # plot od600 + coverage curve
   # od600 should be a np array with replicates
   if fig == '':
      fig = plt.figure(figsize=[3.5, 2])
   ax1 = fig.add_subplot(111)
   ax2 = ax1.twinx()
   #ax1.plot(duration_od600, od600, marker='.', color='blue
      ', label='OD600')
   ax1.errorbar(duration_od600, od600.mean(axis=1), yerr=
      od600.std(axis=1), color='blue', fmt='o', mfc='white',
       linestyle='dashed', label='OD$_{600}$', markersize=6,
       linewidth=2, elinewidth=2, capsize=5)
   ax2.plot(duration_coverage, coverage, marker='.', color='
      red', label='%␣cell␣coverage')
   #ax2.errorbar(duration_coverage,
   ax1.set_xlabel('time␣/␣h')
   ax1.set_ylabel('OD$_{600}$')
   ax2.set_ylabel('%␣cell␣coverage')
   fig.legend(loc=4)
   plt.show()
```

Code A.7 is ran directly given the time point and OD600 points from a given experiment. With a given directory, it will save a .pkl file containing time point, OD600, percent coverage, number of colonies, sizes of colonies, mean and standard deviations of percent coverage, and mean and standard deviations of colony size.

Listing A.7: High-level scripts required for image thresholding.

```
# calculate cell coverage over time of images from multiple
   subdirectories, organizes to df, saves to .pkl
# ensure parameters are correct: direct, duration, od600,
   npics
# subdirectories must have same amount of coverslips
# binning does nothing

import numpy as np
import pandas as pd
import bfanalysis as bf
import os, pickle
import matplotlib.pyplot as plt

def find_subdirects(direct):
   # find unique hours/coverslips from subfolder names (
      assuming 'h_'), grab unique hours and unique coverslip
       reps
   subdirects = []
   files = os.listdir(direct)
   for f in files:
      if f.find('h_') != -1:
         if f.find('.pickle') == -1:
            subdirects.append(f)
   print('␣ENSURE␣subdirectories␣with␣"h_"␣ARE␣IMAGE␣
      SUBDIRECTORIES')

   subdirects_temp2 = []
   subdirects_temp3 = []
   for i in range(len(subdirects)):
      subdirects_temp2.append(subdirects[i].split('_')[0]) #
          select everything before 'h_'
      subdirects_temp3.append(subdirects[i].split('_')[1]) #
          select everything after 'h_'
   hrs_unique = np.unique(np.array(subdirects_temp2))
   coverslips_unique = np.unique(np.array(subdirects_temp3))
```

```
    # from unique finds, sort subdirectories based on
       timepoint and coverslip reps
    subdirects_sorted = []
    for i in range(len(hrs_unique)):
       hrs_sorted = []
       for j in range(len(subdirects)):
          if subdirects[j].find(hrs_unique[i]) == 0:
             hrs_sorted.append(subdirects[j])
       subdirects_sorted.append(hrs_sorted)
    return subdirects_sorted, hrs_unique, coverslips_unique

def df_imgs(direct, subdirects_sorted, coverslips_unique,
    hrs_unique, pxconverter, bins=100):
    # analyze and arranged to dataframe
    # requires uniques
    preview = False
    list_frac_overall = []
    list_counts_overall = []
    for i in range(len(subdirects_sorted)):
       list_frac_tp = []
       list_counts_tp = []
       for j in range(len(subdirects_sorted[i])): # per
          timepoint
          folder = '%s%s/' %(direct, subdirects_sorted[i][j])
          files = os.listdir(folder)
          filenames = []
          for f in files[0:npics]: # grab all .png files to
             be analyzed in a batch
             if f.find('.png') != -1:
                filenames.append('%s%s' %(folder, f))
          list_frac, list_counts = bf.cellcoverage_multiavg(
             filenames, pxconverter, bins, preview) # analyze
              each folder and bin all data in a list #bins
          print('%.2f%% coverage - %s - rep %s' %(np.mean(
             list_frac), hrs_unique[i], coverslips_unique[j])
             )
          list_frac_tp.append(list_frac)
          list_counts_tp.append(list_counts)
       list_frac_overall.append(list_frac_tp)
       list_counts_overall.append(list_counts_tp)
    df_frac = pd.DataFrame(list_frac_overall, columns=
       coverslips_unique, index=hrs_unique) # [index/hrs,
       cols/reps]
    df_counts = pd.DataFrame(list_counts_overall, columns=
```

```
      coverslips_unique, index=hrs_unique)
   return df_frac, df_counts

def df_coverslips(df, coverslips_unique, hrs_unique):
   # calculates mean and st. dev. for all coverslips +
      combined, arranged to dataframe
   list_mean_frac = []
   list_std_frac = []
   for i in range(len(df)):
      mean_rep = []
      std_rep = []
      combined = []
      for j in range(len(df.iloc[i])):
         mean_temp, std_temp = average_list((df.iloc[i, j]))
         mean_rep.append(mean_temp)
         std_rep.append(std_temp)
         combined = combined + df.iloc[i, j]
      mean_temp, std_temp = average_list(combined)
      mean_rep.append(mean_temp)
      std_rep.append(std_temp)
      list_mean_frac.append(mean_rep)
      list_std_frac.append(std_rep)
   df_average = pd.DataFrame(list_mean_frac, columns=np.
      append(coverslips_unique, 'all'), index=hrs_unique)
   df_std = pd.DataFrame(list_std_frac, columns=np.append(
      coverslips_unique, 'all'), index=hrs_unique)
   return df_average, df_std

def average_list(data):
   # calculates mean, st. dev. of a list of numbers
   return np.mean(data), np.std(data)


## MAIN ##
if __name__ == '__main__':
   if os.name == 'posix':
      direct = '/mnt/cluster-victor/lin_motors/pics/raster
         /20190320/' # ENSURE CORRECT PATH TO GRAB DATA AND
         SAVE RESULTS
   else:
      print('␣computer␣is␣not␣linux,␣check.')

   # MUST CHANGE PARAMETERS
# duration = [] #ensure empty or completed
# od600 = [] # ensure empty or completed
```

```
    duration = [0, 3.5, 4.75, 5.5, 6.833, 8.833, 11.25, 23]
    od600 = [0, .22, .62, .91, 1.20, 1.45, 1.72, 2]

    # tier 2 parameters
    saveas = 'data'
    npics = 40 # select how many imgs to analyze per slip
    # do not change (unless different FoV)
    pxconverter = 0.02562347656 # set correct um/px
       resoltuion

    subdirects_sorted, hrs_unique, coverslips_unique =
       find_subdirects(direct) # sort folders
    df_frac, df_counts = df_imgs(direct, subdirects_sorted,
       coverslips_unique, hrs_unique, pxconverter) #
       calculate coverage, organize into df

    df_frac_avg, df_frac_std = df_coverslips(df_frac,
       coverslips_unique, hrs_unique) # calculate avg/stds
    df_counts_avg, df_counts_std = df_coverslips(df_counts,
       coverslips_unique, hrs_unique)

    # ensure length of duration input matches data
    if len(duration) == 0:
       duration = np.linspace(0, 24, len(hrs_unique)).tolist
          ()
       print('␣did␣not␣input␣duration')
    if len(od600) == 0:
       od600 = np.linspace(0, 0, len(duration)).tolist()
       print('␣did␣not␣input␣od600')
    if len(duration) != len(hrs_unique):
       raise Exception('␣inputted␣duration␣!=␣unique␣hours')

    # save to .pkl file
    pickle.dump([[df_frac, df_frac_avg, df_frac_std], [
       df_counts, df_counts_avg, df_counts_std], [duration,
       od600]], open('%s%s.pkl' %(direct, saveas), 'wb'))
elif __name__ != '__main__':
    print('␣growthanalysis_timepoint.py␣imported.')
```

Code A.8 is an example of running code A.7 from the Linux terminal, outputting percent coverage of each image analyzed, and saving a .pkl file in the selected directory.

Listing A.8: Example of running an thresholding on experimental images.

```
dkhlab@dkhlab-scope3:/mnt/cluster-victor/analysis/counting$
```

```
   python growthanalysis_time point.py
4.19% coverage - 0h - rep 1
2.65% coverage - 0h - rep 2
0.10% coverage - 0h - rep 3
0.66% coverage - 0h - rep 4
[...]
29.19% coverage - 24h - rep 1
49.10% coverage - 24h - rep 2
28.27% coverage - 24h - rep 3
34.39% coverage - 24h - rep 4
```

A.4 Spectrometer Control

Code A.9 controls the Acton SpectraPro 2750 monochromator and Princeton Instruments Pixis 400BR CCD array camera. Options to change center wavelength, groove gratings, integration time, number of averages, and plotting and saving spectra are available. It can be imported or ran directly. Running the script directly allows a video preview of spectra being collected. Co-author: Dennis Hore.

Listing A.9: High-level scripts to control Raman spectrometer.

```
#!/usr/bin/env python
# Only takes Pixis images
# slit width: each division is 10 um, with numerical are 1
   mm. min(0.00), max(3.00)m

from scipy import *
import pylab, pvcam, serial, sys, datetime, pickle
from serial import Serial
from scipy.interpolate import interp1d
import numpy as np
import time
import matplotlib.pylab as plt


'''
select_monomove()
get_monomove()
get_wlrange()
initalize_pixis()
class raman()
```

```
   close_pixis()
   change_binning()
   change_integration()
   change_averages()
   change_monomove()
   acquire_pixis()
   acquire_img()
   acquire_spec()
   acquire_plot()
   save_data()
   load_bg()
   plot_img()
   plot_spec()
   preview_live()
'''

def select_monomove(wl_centre=650, no_grating=0):
   # STILL REQUIRES INITILIZE BASH SCRIPT IF TURNED OFF
   # input centre wavelength and grating
   # 1: 1200 gr/mm (bl 700), 2: 600 gr/mm (bl 1000), 3: 300
      gr/mm (bl 1000)
   if wl_centre == 0:
      input_monomove = raw_input('␣input␣no_grating␣&␣
         wl_centre␣(ex.␣690␣3):␣')
      wl_centre, no_grating = input_monomove.split('␣')
   else:
      no_grating = str(no_grating)
      wl_centre = str(wl_centre)
   # setup
   sp = Serial('/dev/ttySP', baudrate=9600, timeout=1) #
      Serial('/dev/ttyUSB0' ...)
   sp.readlines()
   # switch grating/wl
   sp.write('%s␣GRATING\r' %no_grating)
   sp.readlines()
   time.sleep(15)
   # switch centre wavelength
   sp.write('%s␣GOTO\r' %wl_centre)
   sp.readlines()
   # query new settings
   sp.write('?GRATING\r')
   gn = sp.readlines()
   gn = gn[-1].split()[0]
   sp.write('?NM\r')
   centre = sp.readlines()
```

```
    centre = int(float(centre[0].split()[0]) + 0.5)
    sp.close()
    print('␣centre:␣' + str(centre) + ',␣grating:␣' + str(gn)
        )

def get_monomove():
    # see where the monochromator is, to record info
    # gn, grating, centre, wl, wn information
    sp = Serial('/dev/ttySP', baudrate=9600, timeout=1)
    sp.readlines()
    sp.write('?GRATING\r')
    gn = sp.readlines()
    gn = gn[-1].split()[0]
    sp.write('?NM\r')
    centre = sp.readlines()
    centre = int(float(centre[0].split()[0]) + 0.5)
    sp.close()
    if gn == '1':
        grating = '1200␣gr/mm'
        calFile = '/mnt/cluster-victor/lin_cam/1200
            grDispersion.dat'
    elif gn == '2':
        grating = '600␣gr/mm'
        calFile = '/mnt/cluster-victor/lin_cam/600grDispersion
            .dat'
    elif gn == '3':
        grating = '300␣gr/mm'
        calFile = '/mnt/cluster-victor/lin_cam/300grDispersion
            .dat'
    else:
        print('unknown␣grating\n')
        sys.exit(1)
    print('␣grating:␣%s,␣␣centre:␣%s' %(grating, centre)) #
        just to make sure it's correct
    # based on mono setting, figure out the pixel wavelengths
    wl_c, wl_low, wl_high = loadtxt(calFile, unpack=True)
    f = interp1d(wl_c, wl_low)
    x_low = f(centre)
    f = interp1d(wl_c, wl_high)
    x_high = f(centre)
    wl = linspace(x_low, x_high, 1340) # full range of
        wavelengths to plot
    # determine Stokes Raman shift in cm-1
    freq = 10000./(wl/1000.)
    stokes = 10000./.6328 - freq # full range of stokes shift
```

```
        to plot
    return gn, grating, centre, wl, stokes

def get_wlrange(wl_centre, no_grating):
    # get stokes shift range before actually moving
        monochromator
    from scipy.interpolate import interp1d
    gn = str(no_grating)
    centre = int(wl_centre)
    if gn == '1':
        calFile = '/mnt/cluster-victor/lin_cam/1200
            grDispersion.dat'
    if gn == '2':
        calFile = '/mnt/cluster-victor/lin_cam/600grDispersion
            .dat'
    if gn == '3':
        calFile = '/mnt/cluster-victor/lin_cam/300grDispersion
            .dat'
    else:
        print('␣unknown␣grating')
    wl_c, wl_low, wl_high = loadtxt(calFile, unpack=True)
    f = interp1d(wl_c, wl_low)
    x_low = f(centre)
    f = interp1d(wl_c, wl_high)
    x_high = f(centre)
    print('␣%d␣to␣%d␣nm' %(x_low, x_high))
    freq = 10000./(x_low/1000.)
    stokes_low = 10000./.6328 - freq
    freq = 10000./(x_high/1000.)
    stokes_high = 10000./.6328 - freq
    print('␣%d␣to␣%d␣cm-1␣shift' %(stokes_low, stokes_high))

def initialize_pixis(integration=20, averages=1):
    # set up pixis, set exposure time
    a = pvcam.model.DigitalCamera # do you need to make this
        into a variable?
    pixis = pvcam.PVCam(a, 0, 0)
    pixis.exposureTime.value = integration/1000.
    #print pixis.exposureTime.value
    #pixis.resolutionFitter([100,200])
    # set up monochromator information
    print('␣integration:␣%d,␣averages:␣%d' %(integration,
        averages))
    return pixis, a, integration, averages # , gn, grating,
        centre, wl, stokes, averages
```

```
class raman:
    # encloses pixis into raman class
    # includes: close_pixis, binning, acquire_img,
       acquire_spec, save_data, load_bg
    def __init__(self, direct, integration, averages, binning
       =400, bg_subtract=False):
        self.direct = direct # where to load/save data
        self.gn, self.grating, self.centre, self.wl, self.
           stokes = get_monomove()
        self.pixis, self.a_before, self.integration, self.
           averages = initialize_pixis(integration, averages)
        self.change_binning(binning) # does this work?
        self.settings = [self.gn, self.grating, self.centre,
           self.integration, self.averages, self.stokes] # ex.
            [3, 1200, 633, 500, 3, [range]]
        self.bg_subtract = bg_subtract

    def close_pixis(self):
        self.pixis.terminate()
        print('␣pixis␣closed')
        return

    def change_binning(self, binning_value):
        self.pixis._binning = (1, binning_value) # bins x (
           never use), bins y (should either be 1 or 400, what
            happens when between the two?)
        self.binvalue = binning_value
        '''
        if self.binvalue == 400:
            print(' Raman spectrum selected.')
        elif self.binvalue == 1:
            print(' Raman image selected.')
        else:
            print(' not 400 or 1.')
        '''
        return

    def change_integration(self, integration_value):
        self.pixis.exposureTime.value = integration_value
           /1000. # will this work?
        self.integration = integration_value
        print('␣integration␣time:␣%d␣ms' %integration_value)
        return
```

```
def change_averages(self, averages_value):
    self.averages = averages_value
    print( ' averages: %d' %averages_value)
    return

def change_monomove(self, wl_centre, no_grating):
    # change monochromator
    select_monomove(wl_centre, no_grating)
    self.gn, self.grating, self.centre, self.wl, self.
        stokes = get_monomove()
    return

def acquire_pixis(self):
    # NOT USING ATM
    # acquires pixis image/spectrum based on binning.
        Averaging only works for spectra
    #pixis._image_rect = (0, 1339, 0, 399)
    if self.averages == 1:
        im = self.pixis.data.get()
        if self.binvalue == 400: # complete y binning, you
            can just grab directly from within, so don't
            have to do it in later stages
            im = np.array(im[0]) # data in the first item in
                a one-item list(ince all being binned to one
                vector), converted from odemis to numpy
                array
        return im
    else:
        total = zeros(1340)
        for x in range(self.averages):
            im = self.pixis.data.get()
            total += im[0]
            total = np.array(total)
        return total/self.averages

def acquire_img(self, save_img=False):
    # acquire img
    # can't do averages/bg or try converting to array then
        doing averages/bg?
    if self.binvalue == 1:
        im = self.pixis.data.get()
        self.data_img = np.array(im)
        if save_img == True:
            comment = raw_input(' saving img, add comment?')
            self.save_data(comment, is_img=True)
```

```
            elif isinstance(save_spec, basestring):
                self.save_data(save_spec)
        else:
            print('␣error:␣binning␣is␣not␣1.')
        return

    def acquire_spec(self, save_spec=False, is_bg=False):
        # take pixis spectrum, binning must be 400
        # also can acquire bg
        if self.binvalue == 400:
            if self.averages == 1:
                im = self.pixis.data.get()[0] # does this work?
            else:
                total = np.array(zeros(1340))
                for x in range(self.averages):
                    im_temp = self.pixis.data.get()[0]
                    total += np.array(im_temp)
                    im = total / self.averages #np or no np array
                        ?
            im = np.array(im)
            if is_bg == False:
                self.data = im # save variable into self
                if save_spec == True:
                    comment = raw_input('␣saving␣data,␣add␣
                        comment?␣')
                    self.save_data(comment) # default would
                        either be 1 or blank/space
                elif isinstance(save_spec, basestring):
                    self.save_data(save_spec)
            elif is_bg == True:
                self.bg = im # save variable into self
                self.bg_subtract = True
                if save_spec == True:
                    comment = raw_input('␣saving␣bg,␣add␣comment␣
                        [atleast␣add␣bg]?␣')
                    self.save_data(comment)
                elif isinstance(save_spec, basestring):
                    self.save_data(save_spec)
        else:
            print('␣error:␣binning␣is␣not␣400.')
        return

    def acquire_plot(self, save_spec=False, is_bg=False):
        # acquire and plot img or spectrum
        if self.binvalue == 400:
```

```
            self.acquire_spec(save_spec, is_bg)
            self.plot_spec()
        elif self.binvalue == 1:
            self.acquire_img(save_spec)
            self.plot_img()

    def filenaming_raman(self, comment):
        # filename with raman parameters
        filename = '%sd%s_g%s_%snm_%dms_%davgs_%s.pkl' %(self.
            direct,
                datetime.datetime.now().strftime('%Y%m%d-%H%M%S'
                    ),
                self.gn, self.centre, self.integration, self.
                    averages, comment)
        return filename

    def save_data(self, comment, is_img=False):
        # save pixis data and xrange, with spectrometer
            parameters, possible comment as pickle
        filename = self.filenaming_raman(comment)
        '''
        filename = '%sd%s_g%s_%snm_%dms_%davgs_%s.pkl' %(self.
            direct,
                datetime.datetime.now().strftime('%Y%m%d-%H%M%S
                    '),
                self.gn, self.centre, self.integration, self.
                    averages, comment)
        '''
        # pickle version
        if is_img == False:
            pickle.dump([[self.gn, self.centre, self.
                integration, self.averages, self.wl], self.
                stokes, self.data], open(filename, 'wb')) # [
                spectrometer parameters + wl], stokes shift,
                data
        elif is_img == True:
            pickle.dump([[self.gn, self.centre, self.
                integration, self.averages, self.wl], self.
                stokes, self.data_img], open(filename, 'wb')) #
                [spectrometer parameters + wl], stokes shift,
                data
        print('␣%s␣data␣saved.' %filename)
        return

    def load_bg(self, picklename):
```

```
        # import bg spectrum from pickle file
        # ex. d20180915-093025_g3_633nm_500ms_3avgs_comment.
            pkl
        filename = self.direct + picklename
        settings_bg, stokes_bg, data_bg = pickle.load(open(
            filename), 'rb')
       # check if bg is the correct, there should be no
           errors
        if settings_bg[2] != self.integration:
            print('␣error:␣integration␣times␣different.␣
                Intensity␣will␣differ.\n')
        if settings_bg[3] != self.averages:
            print('␣error:␣different␣averages.␣Noise␣will␣
                differ.␣\n')
        if settings_bg[0] != self.gn:
            print('␣error:␣different␣grating.␣Stokes␣range␣will
                ␣differ.\n')
        if settings_bg[1] != self.centre:
            print('␣error:␣different␣centre.␣Stokes␣range␣will␣
                differ.\n')
        self.bg = data_bg
        self.bg_subtract = True
        return

    def plot_img(self, fig=''):
        # plot image usually companied with spectrum
        if self.binvalue != 1:
            print('␣possible␣error:␣bin␣value␣is␣not␣currently␣
                1')
        if fig == '':
            fig = plt.figure() # not required if using
                preview_live()
        ax1 = fig.add_subplot(211)
        ax1.set_yticks(range(0, 401, 100))
        ax1.set_xlabel('wavelength␣/␣nm')
        ax1.set_ylabel('Y␣pixel␣number')
        ax1.imshow(self.data_img, extent=[min(self.wl), max(
            self.wl), 1, 400], aspect='auto')
        plt.show()
        return ax1 # is this required?

    def plot_spec(self, fig='', manual_yrange=False, dual=
        False):
        # plot spectrum
        if self.binvalue != 400:
```

```
            print('␣possible␣error:␣bin␣value␣is␣not␣currently␣
               400')
        if fig == '':
            fig = plt.figure() # not required if using
               preview_live()
        if dual == True:
            ax2 = fig.add_subplot(212) # if want both image and
                spectrum
        elif dual == False:
            ax2 = fig.add_subplot(212) # change this to 111
               with nice graph later?
        ax2.set_xlabel('Stokes␣shift,␣$\Delta␣\omega$␣/␣cm$
           ^{-1}$')
        ax2.set_ylabel('counts')
        ax2.set_xlim(min(self.stokes), max(self.stokes))
        if manual_yrange == True:
            ylim_low = raw_input('␣ylim␣low?:␣')
            ylim_low = int(ylim_low)
            ylim_high = raw_input('␣ylim␣high?:␣')
            ylim_high = int(ylim_high)
        elif manual_yrange == False:
            ylim_low = min(self.data)
            ylim_high = max(self.data)
        else:
            ylim_low = manual_yrange[0]
            ylim_high = manual_yrange[1]
        ax2.set_ylim([ylim_low, ylim_high])
        if self.bg_subtract == False:
            ax2.plot(self.stokes, self.data)
        elif self.bg_subtract == True:
            ax2.plot(self.stokes, self.data - self.bg) # does
               bg subtract work?
        ax2.set_title('%s,␣%d␣nm,␣%d␣ms,␣%d␣avgs,␣%d␣max' %(
           self.grating, self.centre, self.integration, self.
           averages, max(self.data)))
        plt.show()
        return ax2

    def preview_live(self, dual=False, manual_yrange=False):
        # preview img+spectrum, or just spectrum. No save
        # preview sometimes gives core dumped sometimes
        pylab.ion()
        pylab.rc('font', size=9)
        #fig = pylab.figure(figsize=(3.25, 8))
        fig = pylab.figure()
```

```
fig.set_tight_layout(True)
try:
   while True:
      #a = time.time()
      if dual == True:
         # acquire + plot image
         pylab.clf() # clear fig?
         self.change_binning(1)
         self.acquire_img(save_img=False)
         ax1 = self.plot_img(fig=fig)
         self.change_binning(400)
      # acquire + plot spectrum
      self.acquire_spec(save_spec=False, is_bg=False)
      if self.bg_subtract == True:
         self.data2 = self.data - self.bg

      ax2 = self.plot_spec(fig=fig, dual=dual,
         manual_yrange=manual_yrange)

      ax2.figure.canvas.draw()
      ax2.clear()

      #b = time.time()
      #print b - a

except KeyboardInterrupt:
   if dual == True:
      ax1 = self.plot_img(fig=fig)
   ax2.clear()
   ax2 = self.plot_spec(fig=fig, dual=dual,
      manual_yrange=False)
   ax2.figure.canvas.draw()
   savefig = raw_input('\n save fig w/ descr. (ensure
      no overwrite): ')
   if savefig != '':
      # saves if input something
      filename = '%sd%s_g%s_%snm_%dms_%davgs_%s.png'
         %(self.direct,
            datetime.datetime.now().strftime('%Y%m%d-%
               H%M%S'),
            self.gn, self.centre, self.integration,
               self.averages, savefig)
      pylab.savefig(filename)
      print(' saved figure.')
      comment = raw_input(' saving spectrum, add
```

```
                comment?␣')
            self.save_data(comment)
            comment = raw_input('␣saving␣img,␣add␣comment?␣'
                )
            self.save_data(comment, is_img=True)
        return


### MAIN CODE ###
# init
direct = '/mnt/cluster-victor/lin_cam/new_data/'
binning = 400 # for spectrum analysis

# initial parameters
if raw_input('␣change␣grating,␣centre␣wl␣(y/n)?␣') == 'y':
    select_monomove(0) # only do if settings are different
if raw_input('␣change␣integration,␣averages␣(y/n)?␣') == 'y'
    :
    input1 = raw_input('␣input␣integration␣averages␣(ex.␣500␣
        1):␣')
    integration, averages = input1.split('␣')
    integration = int(integration)
    averages = int(averages)
else:
    integration = 500
    averages = 1

# placeholder for testing
if raw_input('␣initialize␣raman␣class␣(y/n)?␣') == 'y':
    raman = raman(direct, integration, averages, binning)
    raman.preview_live(dual=False, manual_yrange=False)


# popular functions
#raman.acquire_plot() # along with acuire_spec(), plot_spec
    ()
#raman.preview_live(dual=False, manual_yrange=False)
#raman.close_pixis() # always have at the end
```

Code A.10 is an example of running code A.9 from the Linux terminal, allowing inputs of groove grating, center wavelength, integration time, number of averages, with an option to save the last frame of spectra before closing the spectrometer. $initialize.sh$ to may be

required to set USB permissions properly.

Listing A.10: Example of controlling Raman spectrometer.

```
dkhlab@dkhlab-scope3:/mnt/cluster-victor/lin_cam$ .
   initialize.sh
/home/dkhlab/Downloads/pvcam-pilk-master/usb/rspiusb.ko:
   File exists
dkhlab@dkhlab-scope3:/mnt/cluster-victor/lin_cam$ python
   ramancontrol.py
 change grating, centre wl (y/n)? y
 input no_grating & wl_centre (ex. 690 3): 690 3
 centre: 690, grating: 3
 change integration, averages (y/n)? y
 input integration averages (ex. 500 1): 1000 3
 initialize raman class (y/n)? y
 grating: 600 gr/mm, centre: 690
 integration: 1000, averages: 3
 save fig w/ descr. (ensure no overwrite):
```