processes

MDPI

Article

Virtual Test Beds for Image-Based Control Simulations Using Blender

Akkarakaran Francis Leonard , Govanni Gjonaj, Minhazur Rahman and Helen E. Durand *

Department of Chemical Engineering and Materials Science, Wayne State University, 42 W Warren Ave, Detroit, MI 48202, USA
* Correspondence: helen.durand@wayne.edu

Abstract: Process systems engineering research often utilizes virtual testbeds consisting of physics-based process models. As machine learning and image processing become more relevant sensing frameworks for control, it becomes important to address how process systems engineers can research the development of control and analysis frameworks that utilize images of physical processes. One method for achieving this is to develop experimental systems; another is to use software that integrates the visualization of systems, as well as modeling of the physics, such as three-dimensional graphics software. The prior work in our group analyzed image-based control for the small-scale example of level in a tank and hinted at some of its potential extensions, using Blender as the graphics software and programming the physics of the tank level via the Python programming interface. The present work focuses on exploring more practical applications of image-based control. Specifically, in this work, we first utilize Blender to demonstrate how a process like zinc flotation, where images of the froth can play a key role in assessing the quality of the process, can be modeled in graphics software through the integration of visualization and programming of the process physics. Then, we demonstrate the use of Blender for testing image-based controllers applied to two other processes: (1) control of the stochastic motion of a nanorod as a precursor simulation toward image-based control of colloidal self-assembly using a virtual testbed; and (2) controller updates based on environment recognition to modify the controller behavior in the presence of different levels of sunlight to reduce the impacts of environmental disturbances on the controller performance. Throughout, we discuss both the setup used in Blender for these systems, as well as some of the features when utilizing Blender for such simulations, including highlighting cases where non-physical parameters of the graphics software would need to be assumed or tuned to the needs of a given process for the testbed simulation. These studies highlight benefits and limitations of this framework as a testbed for image-based controllers and discuss how it can be used to derive insights on image-based control functionality without the development of an experimental testbed.

Keywords: process control; image-based control; digital twin

Citation: Leonard, A.F.; Gjonaj, G.; Rahman, M.; Durand, H.E. Virtual Test Beds for Image-Based Control Simulations Using Blender. *Processes* **2024**, *12*, 279. https://doi.org/10.3390/pr12020279

Academic Editors: Maurício Bezerra De Souza, Jr. and Anthony Rossiter

Received: 5 September 2023
Revised: 29 December 2023
Accepted: 4 January 2024
Published: 27 January 2024

1. Introduction

Image-based control (IBC) focuses on the use of image sensors to capture state information for the purposes of control. IBC systems are feedback control schemes in which the measurements are obtained from data-intensive image-based sensors. The advances and emergence of affordable camera sensors and image-processing algorithms [1] have contributed to the increased exploration of IBC systems [2]. This includes applications in autonomous driving systems [3], robot manipulators [4], and bioprocess monitoring [5,6]. Images have also been used in drug discovery [7]. The sensors in many process control applications measure a single state of the system, such as temperature or pressure. Images are able to capture significant information in a single observation, and offer the possibility of creating a dense and relatively easy method to capture system states. Over the last few decades, significant advances in image-based sensors and image processing algorithms have been reported. This has led to an increase in studies of IBC systems and their capabilities to enable real-time process control based on camera sensors to perceive the environment and measure variables that would otherwise be impractical or time-expensive to measure (e.g., concentration grade or cell counting). One of the uses of image sensors is object detection. Images have found use in robotics, in applications such as the control of autonomous robots [8,9], drone flight control [10], manipulation of surgical robots [11], and spatial exploration with mobile devices [12]. In addition to classical controllers, vision-based control strategies have also been studied using advanced controllers such as model predictive controllers (MPC's) [13,14]. In particular, image-based visual servoing control using a nonlinear MPC was proposed in Lee et al. [13] as a vision-based obstacle avoidance strategy in a dynamic environment for an unmanned aerial vehicle, where constraints related to actuator and visibility limitations can be added to the control formulation. In the context of visual feedback control, different control architectures integrated with a vision system (e.g., a fixed camera sensor) have been utilized to improve the performance of autonomous systems [4,15]. In Su and Zheng [4], for example, an image-based transpose Jacobian proportional-integral-derivative (PID) control was proposed to address the asymptotic regulation problem of robot manipulators with a vision-based feedback using Lyapunov's direct method and LaSalle's invariance theorem. In chemical engineering, IBC implementation is largely applied to physical models and relies on experimentation to deliver results and test optimal parameters. Image analysis systems were tested, for example, to monitor an industrial boiler system [16] or estimate bubble size at a phosphorus oxide flotation process [17]. The analysis of images for liquid crystal responses for sensing purposes was explored in [18]. The use of convolutional neural networks (CNNs) to analyze images for real time control has been used to develop frameworks for the safe implementation of computer vision in process control [19] and to extract geometrical and pattern information from images [20]. Image sensors have been evaluated for crystal size monitoring [21,22]. Specialized probes and lighting conditions can aid such studies. Drop size distributions monitoring aimed to use image sensors [23]. In the work by Chen et al. [24], image sensors are used to control combustion, using a principle component analysis (PCA)-based model of flame images. Similar to PCA, dynamic mode decomposition (DMD) is also used to create low-dimensional state representations of images that can be used in conjunction with model predictive control (MPC) [25]. Although image-based closed-loop systems have been performed and tested with real systems involving camera sensors [24,26], chemical processes are often large-scale and complex, making it challenging to visually replicate a next-generation manufacturing environment for the process industries without obtaining data from an actual plant. In light of this, it can be more difficult for process systems engineering researchers to test how new next-generation manufacturing concepts such as image-based control algorithms might fare in an actual plant. A simulation-based test environment for these systems would be desirable. Blender is a free and open source 3D modeling and animation software that is managed by the Blender Foundation, with the aim of providing the latest in animation technology to everyone free of charge and with the ability for creators to own the projects that they create with it. It has been used to create open source repositories for models of laboratory equipment [27], plugins for molecular dynamics simulations [28], and toolboxes for the simulation of range scanners [29] and has been used to develop models for packed beds [30].

In the prior work published by our group [31], we proposed the use of Blender for enabling the generation of images as part of a closed-loop image-based control test paradigm. In that work, a simple example of level control in a tank was used to demonstrate the use of Blender for closed-loop image-based control simulations. In this study, we significantly extend this prior work to seek to gain further insight into the potential use cases of Blender for chemical process control applications and also to better understand its benefits and limitations over a wider range of process considerations. To do this, the paper is structured to focus on three primary studies, which are the use of Blender in simulating a precursor to a froth flotation process, a precursor to a controlled self-assembly process, and a reactor for a process with a weather monitoring system. The specific organization is as follows: in the preliminaries, the method that is used in this work to acquire an optimal policy for the self-assembly precursor system (which is a stochastic system), Bellman optimization, and the basics of modeling objects and animation basics in Blender are covered. The sections following the preliminaries then discuss the simulations carried out to illustrate techniques to replicate physical phenomena and IBC systems in Blender. First, the precursor to the froth flotation process is analyzed to focus on how the animation acquired from Blender compares to results from a numerical integration technique to ensure that physical phenomena are represented with an acceptable degree of accuracy. The precursor simulation in this case is the simplified motion of a gas bubble in a liquid. Second, the precursor simulation to controlled self-assembly is presented to discuss the use of images to detect states that may be desired to be monitored using images as part of a closed-loop process. The precursor simulation in this case is the Brownian motion of a nanorod, simulated in a 2-dimensional space, with the aim of developing a control policy that guides the rod to the center of the simulated space. The control policy is optimized offline and the virtual camera in Blender is used to capture images every sampling period, detect the position and orientation of the nanorod, and perform the optimal action. The simulation functions as a precursor to more complex problems, such as those that involve controlling multiple particles in a medium, as in colloidal self-assembly. Finally, an outdoor tank reactor is simulated that is subject to sunlight as a disturbance in ambient conditions. This simulation describes one of the more practical uses of image sensors for disturbances, since weather effects might be identified through their effects on other system parameters. Camera sensors can enhance the responsiveness of a control system by directly utilizing image information to predict probable impacts on the process due to weather. The optimizer used to control this system includes a disturbance model that accounts for the varying sunlight, which correlates information from the image in order to update the control action. The performance of this optimizer is compared to one that does not possess a similar disturbance model.

Our main goal in this work is to analyze the capabilities of Blender for process engineering applications and to explore its benefits and limitations with respect to such applications and also to discuss how simulations that integrate the animation and visualization capabilities of Blender with more typical process engineering considerations (e.g., numerical integration and control) might be developed. The goal is not to imply that Blender is necessarily the best tool for such simulations (though we view its open-source nature as a positive characteristic for research) but rather to discuss how such simulations might be set up and developed and some of the considerations that must be taken into account in building engineering-relevant simulations within it.

2. Preliminaries

2.1. Notation

$\mathbb{R}$ and $\mathbb{Z}^+$ represent the sets of real numbers and non-negative integers, respectively. The variables denoted in bold (e.g., $\mathbf{r}$, $\mathbf{D}_t$) are tensors. The random variables described in this work are normally distributed. For a random variable x, the mean is denoted by $\langle x \rangle$, and the variance is denoted by $\langle x, x \rangle$. For a normally distributed vector $\mathbf{r}$, the variance is denoted with a transpose as $\langle \mathbf{r}, \mathbf{r}^\mathsf{T} \rangle$.

2.2. Stochastic Processes and Their Control

One of the examples of the use of Blender for integrated process modeling and visualization toward image-based control testing that is presented in this work involves stochastic processes. We, therefore, highlight the key background on stochastic processes and their control that comes up in the discussion of that system.

Stochastic processes are defined with random variables, leading to a non-deterministic behavior [32,33]. A process is said to have the Markov property if the probability of a future state is equivalent regardless of whether all the previous states are known or if only the current state is known. A Markov decision process (MDP) is a subset of stochastic processes based on Markov chains, where the probability of a future state of a process is dependent on its current state. MDPs modify this definition by introducing a choice of action to be taken, i.e., the probability of a given future state is dependent on both the current state and the action taken in that state. A MDP is defined as a tuple of 4 sets: S, the state space of the system; A, the set of actions available for control; P, the transition probabilities; and R, the immediate reward of achieving a given state [34]. The reward set R is generally used to define MDPs in the context of robotics, but in the chemical engineering applications in this work, it is recast as a cost set C.

The control of an MDP is carried out with policy $\pi : S \mapsto A$, which returns an action to be executed when the process occupies a given state. The optimal policy π^* is the one that minimizes the infinite horizon objective function $J(\pi)$, defined as:

$$J(\pi^*) = \min\left[E\left\{\sum_{t=0}^{\infty}\gamma^t C(s_t)\right\}\right] \tag{1}$$

Dynamic programming can be applied to optimize this problem, and in this work, Bellman's optimality condition is used to arrive at an optimal policy. The value function $V : S \mapsto \mathbb{R}$ of state s is a representation of the cost of the current state as well as the cost of any probable future states that can be accessed from the current one. The Bellman optimality condition defines the optimal value function V^* for an MDP recursively, as in Equation (2) [35]:

$$V^*(s) = C(s) + \gamma \min_{a\in A} \sum_{s'\in S} P(s'|s,a)V^*(s') \tag{2}$$

A greedy algorithm can attempt to minimize the cost associated with the next action. The transition probability $P(s'|s,a)$ in Equation (2) captures the non-determinism of the system by weighing the values $V^*(s')$ of future states s' by the probability of acquiring that future state, given the current state s and the action a taken at the time. S represents the set of all possible states of the system. The discount factor γ penalizes the value of future states by adding a fraction of the future states to the current one.

Equation (2) is recast as an update rule to perform value iteration, as in Equation (3), known as the Bellman update rule [35]:

$$V_{i+1}(s) = C(s) + \gamma \min_{a\in A} \sum_{s'\in S} P(s'|s,a\ )V_i(s') \tag{3}$$

Here, i represents the index of iteration. Through repeated iterations, the value of every state in the system is converged, resulting in an optimal value function $V^*(s)$. This algorithm presented here and used in this work is called value iteration and is one of several algorithms used to optimize MDPs. The mapping of the value function from a previous iteration V_i to the next V_{i+1} is a contraction mapping, referred to as the Bellman operator [35]. The action selected to minimize the value (and by extension the cost) forms the policy of the system. Unlike the policy of deterministic systems that are trajectories through time, the optimal policy $\pi^* : S \mapsto A$ of stochastically controlled systems consists of optimal actions connected to a given state of the system.

2.3. Modeling and Animation in Blender

Blender is a computer graphics software that is able to represent the three-dimensional world spatially, represent it in action (i.e., animated), and interface such actions on the visual world with codes through a Python coding interface that can be used to code typical modeling strategies in chemical engineering (e.g., control laws or process physics). Blender is an open-source program that can be downloaded for free and is licensed under the GNU General Public License. This means that projects created with Blender are the sole property of the user, which makes it of interest to evaluate for potential research purposes. In Blender, three-dimensional worlds can be constructed by adding basic shapes (e.g., cylinders and icospheres) and then, from there, performing operations (e.g., extrusion, scaling, translating) to make them into models of various objects in three-dimensional space. The movement of these objects can then be added, either by adding rigs to achieve allowable motions of the object to give it more complex movements, or through, as is performed in any cases in which animation is used in this work, translations, rotations, and scalings of the original object throughout space. This section goes over the features of Blender that make modeling possible and how they can be leveraged to simulate target processes. Figure 1 details the information flow of the simulations described in this paper. In each of the simulations, the objects in the 3D environment, materials, lighting, and environment are initially designed in Blender and stored in a .blend file. Then, a Python script (.py script) is used to encode the process dynamics, animation of object properties, and image capture using the virtual camera. These images are stored in a local directory, then loaded back into the script using Pillow [36], which is then used to analyze the image data in order to perform state detection. Additionally, parameters of the process such as calibration information, control policies that are optimized offline, or other physical parameters can be loaded from the local directory through the Python script. In this work, the parameters used for the simulation are stored as .mat files and are loaded into the simulation using the `savemat` function, available in the SciPy library. After the script is run, the results are stored in .mat files, and the video of the animation, as well as images from the process, may be rendered from the .blend file.

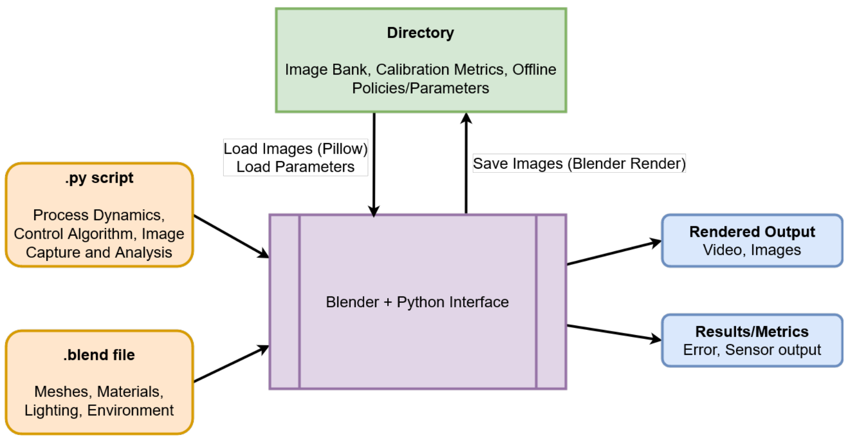
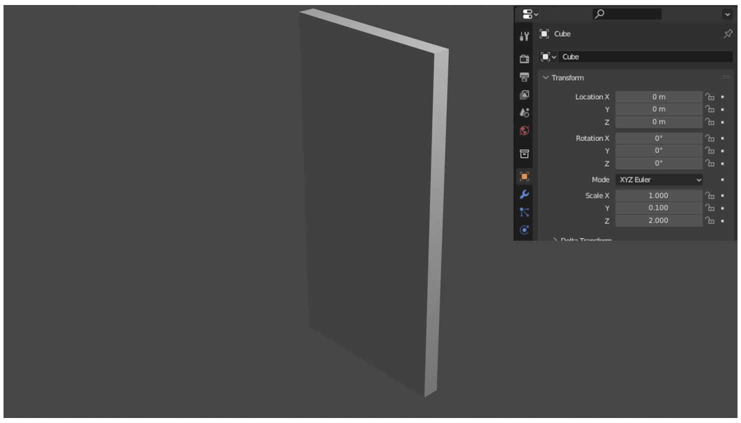

Directory
Image Bank, Calibration Metrics, Offline Policies/Parameters
Load Images (Pillow)
Load Parameters
Save Images (Blender Render)
.py script
Process Dynamics, Control Algorithm, Image Capture and Analysis
.blend file
Meshes, Materials, Lighting, Environment
Blender + Python Interface
Rendered Output
Video, Images
Results/Metrics
Error, Sensor output

Figure 1. Information flow for IBC simulations.

2.3.1. Object Creation and Sizing

Most three- and two-dimensional objects in Blender are categorized as "meshes" and can be added to a scene. When an object is added, it appears at the location of the cursor occupying the space of a 2 m $\times$ 2 m $\times$ 2 m cube. The location, size, and orientation of an object can be found by selecting the object in the 3D viewport and navigating to the object properties tab. The location of a new object can be specified relative to the location of the cursor in meters, and its orientation can also be modified. The rotation of objects in Blender is determined in Euler mode by default but can be changed to Quarternion or Axis Angle mode. The scaling of objects in Blender is performed relative to the initial size and, hence, does not use units. For example, to define a cuboid of size 2 m $\times$ 20 cm $\times$ 4 m, the scale factors in the X, Y, and Z directions are set to 1, 0.1, and 2, respectively, from a base-case cube that has been added to the scene. When implemented, the rendered output would look like the image displayed in Figure 2, with the inset displaying the scale factors. This implies that when setting physical parameters, a scaling factor relative to how objects are scaled in the simulation must be included. By default, a newly created object is not equipped with a material and, hence, appears white in the 3D viewport window when the solid shading option is selected. Selecting an object and navigating to the material properties tab, the object can be equipped with a material, which allows the specification of material properties like color and roughness. These properties specify how an object appears in rendered images and animations and can be previewed by changing the shading in the 3D Viewport to material preview or rendered.



Location X	0 m
Y	0 m
Z	0 m
Rotation X	0°
Y	0°
Z	0°
Mode	XYZ Euler
Scale X	1.000
Y	0.100
Z	2.000

Figure 2. Rendered output of rescaled cube, with inset added to show scale factors.

2.3.2. Object Property Animation Using Blender's Python Interpreter

In order to animate a process, images are rapidly replaced in order to produce the illusion of motion. These images are called frames, and the rate of replacement of frames determines how smooth the animation appears to a viewer, termed as frames per second (FPS). For an object translating from an initial to a final position, each frame captures the intermediate position of the object as it moves. Instead of specifying what intermediate position the object should occupy at each frame, only the initial and final positions are specified at their corresponding frames in Blender. When the position, rotation, or scaling of an object is specified in a frame, the frame is called a keyframe. Blender interpolates the intermediate object properties between keyframes. The number of frames between keyframes is determined by the time interval between keyframes and the FPS. In the projects described in the following sections, keyframes are generated using Python scripts after determining the total runtime for a simulation. Blender includes a version of Python, and many commonly used Python packages can be installed. Script files may be stored within the project file (.blend file), or locally as a Python script file (.py file) that can be loaded into the project and edited outside of Blender. Most modifiable properties in the Blender interface possess a Python data path that can be acquired by enabling Python Tooltips in the Preferences menu in Blender. Using the corresponding data path of a property, Python scripts can be programmed to modify object properties, insert keyframes, acquire data from intermediate frames, and render still images to simulate IBC processes. Using Blender's Python interpreter, any property that can be accessed in the Properties window can be animated in this way.

2.3.3. State Detection Using Images

Animations in Blender are captured and rendered through a virtual camera, which can be manipulated like any other object in the environment (e.g., it can be placed in different locations in the scene or rotated). The camera can also be used to render images during the running of Python scripts used to simulate processes. To load these images during the runtime of the script in order to analyze them, Python libraries that provide this capability can be loaded. In order to carry this out for image-based state detection in this work, a fork of the Python Image Library, Pillow [36], is used to load images for analysis. This library allows us to isolate the different color channels, cast them into arrays, and carry out the necessary algorithms in order to carry out state detection. Setting surroundings and lighting for objects in Blender can aid in state detection from images, as well as bring simulations closer to representing physical processes. Light sources in Blender come in 4 types: Point, Sun, Spotlight, and Area. Depending on the light source being simulated, these 4 types can be modified, for example, by changing color and intensity, to more closely simulate a target process. Material properties also interact with lighting to produce reflections and glare in the simulated process, which may or may not be desirable depending on the use case. With these fundamentals in hand, the following sections detail Blender's capabilities in the context of several pathfinding examples for chemical process simulation.

3. Use of Blender for Modeling and Image-Based Control Tests of Self-Assembly and Environment Recognition

Our prior work [31,37,38] provided a proof of concept that due to the properties of animation, modeling, and coding available in Blender as described in Section 2.3, Blender is a potentially viable option for simulating and testing image-based control laws. However, despite the preliminary analyses of its capabilities as presented in Oyama et al. [31,37,38], many aspects of its performance and interaction with the development of image-based control testbeds remain to be explored and understood for chemical processes to provide a more comprehensive guide to how to successfully develop an image-based control simulation for evaluation purposes in Blender. For example, 3D modeling and animation in Blender abstracts information from the user to make it more friendly for artists, including interpolating movement during the animation process and using non-physical parameters in aspects such as lighting source selection. For the purposes of replicating a chemical process, access to some of this information and determining how to (and when to) manipulate it in a useful manner are key to using Blender as a simulation tool. This section uses three process examples that are precursors to more relevant image-based control simulations than the tank level example in [31,37,38] to explore aspects of how one might work with Blender's capabilities to analyze whether an image-based control simulation has an appropriate setup. Specifically, the following simulations are used for these analyses. First, a simulation of bubbles moving upward (a precursor to modeling their upward movement through a fluid as part of moving toward the modeling of industrial processes such as mineral flotation) is presented. Through this simulation, we analyze the extent to which allowing Blender to interpolate motion between frames where positions and rotations are specified for animation impacts the trajectory of the object compared to hard-coding the information more frequently. The interpolated states are acquired by capturing frame data and are compared with results from the numerical integration of the system model. Second, a nanorod moving stochastically in a two-dimensional plane directed by the actions of an on–off controller that is actuated based on image sensing is presented (considered to be a precursor toward simulations of self-assembly in [39]). In this simulation, we demonstrate how Blender can be utilized to identify challenges in properly sensing objects in image-based control, such as the color of the background being difficult to distinguish from the color of the object such that image processing may need to be more sophisticated to locate an object to which it is desired to apply an optimal control action. Finally, Blender is used to model sunlight shining with different strengths on a reactor, which is then mapped to different heat levels impacting the equipment. Thus, Blender is used to model environmental disturbances to a system, which are then identified through the image-based sensors and subsequently compensated. We discuss how one might integrate Blender's lighting models with non-physical parameters with process modeling and build toward more comprehensive disturbance-handling in control through the use of image sensors that can help to identify causes of changes in the process operation. We compare the responses of the closed-loop system with and without the extra information available from the image capture system to showcase the potential benefits of using Blender for evaluating disturbance-handling techniques. This series of examples provides a deeper look at the potential uses of Blender (and other similar graphics software) toward process systems engineering use cases and how to analyze its benefits and limitations and overcome limitations toward developing closed-loop image-based control tests.

3.1. Animation of Bubble Motion in Blender

This section presents our first demonstration of the use of Blender toward more realistic image-based control applications than were outlined in [31,37,38], where only monitoring and control of tank level using a camera sensor was considered. In those simulations, the tank level was only modeled in two dimensions (i.e., the level was modeled as a plane, similar to drawings that might be made during a controls class but not representative of a physical tank with multiple dimensions), and furthermore, processes such as tank level often have robust measurement techniques through, for example, traditional level sensors that make it less likely for image-based sensing to be used. In contrast, this section focuses on modeling bubble motion in Blender, which is considered to be a precursor to modeling a froth flotation process that could aid in facilitating tests of image-based controllers for a process such as zinc flotation. Image analysis has been previously considered for processes such as zinc flotation and studied using experimental systems [40–42]. In general, image analysis can play an important role in industrial processing [1,43], and this section demonstrates how to model systems in Blender toward representing relevant industrial processes.

Chemical engineering processes are generally described with systems of differential equations that are integrated numerically. The results from numerical integration are a collection of discrete state and time pairs, which can be used in model-based control as predictions of future process conditions and thereby aid in selecting control actions. In Blender, keyframes function as analogues to these discrete state–time pairs, and the software creates intermediate frames by interpolating between the states. In this section, Blender is used to animate the motion of an air bubble through water and determine how much the interpolated states deviate from a numerically integrated result. The equations of motion of a gas bubble in a fluid are modeled using Python to acquire the position of the bubble over time. Some of the resulting state–time pairs are then used to set the keyframes in Blender (i.e., to set the bubble location in these keyframes). The trajectory of the bubbles resulting from the full set of state–time pairs obtained from numerical integration is then compared with the results of using Blender with a subset of those state–time pairs for setting keyframes and the remaining positions interpolated at times between keyframes.

The process of adding animation to objects in Blender involves inserting keyframes corresponding to specific times (frames) in the simulation. A frame rate must be specified in Blender; using that frame rate, one can compute the time corresponding to a given frame so that specific times of visual events can be coded into Blender. Keyframes are differentiated from regular frames by ensuring that a property of an object in Blender (e.g., its position or rotation) meets a designated value when the frame is displayed. Though it is possible to specify properties of an object at every frame, this adds effort to the process of animating the scene due to the high level of specificity required at every frame. Instead, it may be desirable to add keyframes at every several frames instead of every frame. When this is done, Blender interpolates object properties between the keyframes. This creates a smoothing effect between keyframes to give an appearance of smooth motion to a viewer of the animation. However, if the distance between keyframes is large, the interpolation that Blender utilizes may not represent the path that an engineer expects the object to take. Thus, one of the important considerations for developing closed-loop simulations in Blender is determining how often to place keyframes to trade off between specifying the details of the simulation at every frame timescale versus allowing Blender to perform some interpolations. For example, for 24 frames per second, an object's property could be updated every frame (i.e., every 1/24 s) using values of the position from numerical integration with a 1/24 s integration step, though a keyframe might be inserted once every second, i.e., every 24th frame. This would ensure that at the start of every second, the object property would match the value that it would have from integrating the dynamic model of the system. However, over the 24 frames in between the keyframes, the value of the object's properties (e.g., position) would be interpolated if a video of the full scene was rendered from Blender (i.e., they may not be exactly the values expected from the numerical integration). This section uses the bubble modeling process to showcase how the difference between frames and keyframes can lead to some differences in the trajectories compared to the results of using numerical integration.

3.1.1. Equations of Motion of a Bubble

In order to simulate the motion of the bubble, the dynamics of the system are first acquired from the equations of motion [44]. The motion of gas bubbles in a liquid is the result of a number of forces. Since this simulation is intended to function as a pathfinding simulation for the use of graphics toward image-based control applications, the model used in this case is simplified and only accounts for two possible forces on the bubble. Specifically, only the net buoyant force (F_b) and drag force (F_d) are included. The net buoyant force in the upward direction is described in Equation (4), where g is the gravitational acceleration constant, V_{bub} is the volume of the gas bubble, ρ represents the density, and the subscripts liq and bub refer to liquid and bubble properties, respectively:

$$F_b = gV_{bub}(\rho_{liq} - \rho_{bub}) \tag{4}$$

The drag force experienced by a gas bubble is dependent on the flow of the liquid around the gas bubble, characterized by the Reynold's number (Re_{bub}). For low bubble velocities near the initial conditions, the drag force (F_d) is described by Stokes' Law for low Reynold's number, i.e., $Re_{bub} < 1.0$. At higher bubble velocities, the flow around the bubble can become turbulent, which results in a higher Reynold's Number, i.e., $Re_{bub} \geq 1.0$. The drag force in these cases is described by Equation (6), where C_d is the coefficient of drag, which is dependent on the shape and size of the bubble (here, $C_d = 0.2$), r_{bub} is the radius of the spherical bubble, A_{bub} is its surface area, and v_{bub} is the speed of the bubble [45]:

$$Re_{bub} = \frac{2r_{bub}\rho_{liq}v_{bub}}{\eta_{liq}} \tag{5}$$

$$F_d = \begin{cases} -6\pi\eta_{liq}r_{bub}v_{bub} & Re_{bub} < 1.0 \\ -0.5C_d A_{bub}\rho_{bub}v_{bub}^2 & Re_{bub} \geq 1.0 \end{cases} \tag{6}$$

The diffusion of the gas within the bubble to the liquid is assumed to be negligible for this simulation, and hence, the bubble size remains unchanged as it moves through the liquid. With these equations and Newton's third law of motion, the velocity of the bubble is related to the net force it experiences, as in Equation (7), where $v_{bub}(0)$ represents the initial velocity of the bubble and m_{bub} represents its mass:

$$\frac{dv_{bub}}{dt} = \frac{1}{m_{bub}}(F_b + F_d) \quad ; \quad v_{bub}(0) = 0\ \mathrm{m/s} \tag{7}$$

$$x(t + \Delta t) = x(t) + v_{bub}(t) \cdot \Delta t \quad ; \quad x(0) = 0 \tag{8}$$

The position (x) of the bubble is acquired from the numerically integrated velocity, as in Equation (8), where $x(0)$ represents the initial position of the bubble, and Δt represents the integration step using the explicit Euler numerical integration method.

Remark 1. *Cleary et al. [45] describe a number of phenomena involved in more visually accurately modeling bubbles and frothing in graphics. These phenomena include modeling both the fluid and bubbles as separate and unique particles, where the liquid particles carry a certain amount of gas within them, and bubbles are generated whenever enough gas has been concentrated within the liquid particle. The bubbles also interact with both each other and the walls to simulate collisions, and cohesion is accounted for at the liquid surface. Fluid simulation to interact with the bubbles may be carried out using, for example, smoothed particle hydrodynamics principles.*

3.1.2. Animation of Bubble Position in Blender

Equations (7) and (8) are integrated with the parameters listed in Table 1. The radius of the bubbles was chosen to be 2.5 mm. The time and corresponding position values from the numerical integration were stored in two arrays (one for time and one for position). These position and time arrays are imported into Blender in order to animate the motion of the bubble. Three bubbles at different initial positions are placed in a tank. The z coordinates of the bubbles are updated with the integrated displacements, except that the position changes are scaled by a factor of 0.005 to avoid having the bubbles leave the camera view for visualization purposes. In order to animate the bubbles moving, each bubble's location is updated with keyframes, as described in Section 2.3.2. A keyframe is inserted at the initial frame and at every second (i.e., every 24th frame) after until the end of the simulation period. In order to test if the interpolation through intermediate frames is approximately equal to the numerically integrated result, Blender's Python interpreter is used to capture the location of each bubble for all frames of the simulation. This location data is then compared to the position of the bubbles as acquired from integrating Equations (7) and (8). A selection of keyframes from the Blender environment's 3D Viewport is shown in Figure 3.

Table 1. Parameters for bubble motion.

Parameter	Value
Radius of bubble, r_{bub}	2.5×10^{-3} m
Density of liquid, ρ_{liq}	1000 kg/m^3
Viscosity of liquid, η_{liq}	10^{-3} Pa · s
Density of bubble, ρ_{bub}	1.22 kg/m^3
Integration time step, Δt	1/1440 s
Simulation time	120 s

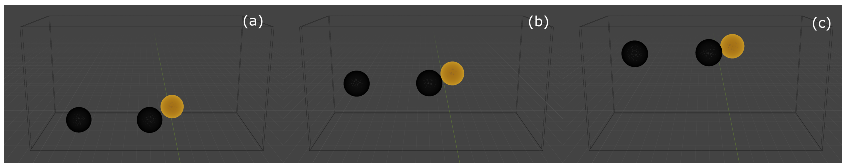


Figure 3. Wireframe captures of the simulation for selected frames: (**a**) frame 0, (**b**) frame 1600, and (**c**) frame 2859. The displacements are scaled by a factor of 0.005. The size of the bubbles are exaggerated for visual identification. The starting positions for each bubble, from left to right, are 1.25885 m, 1.44076 m, 1.30192 m respectively.

3.1.3. Comparing Predictions with Between-Keyframe Interpolations

Because Blender interpolates between keyframes, and since the bubble is given a different position in 3D space at two different keyframes, Blender determines a path between those positions between the two keyframes. This path that it determines may not be equivalent to the numerically integrated path previously described. Therefore, as part of determining how relevant the Blender model is to modeling of the process physics, it is valuable to compare the difference between the numerically integrated path and the path extracted from the simulation including interpolation. To compare the values interpolated by Blender between keyframes and the results of the integration carried out in Python, the displacement of the bubbles in the frames between keyframes is required. This is carried out in Blender by setting up a "for" loop that requests the average displacement (compared to the bubble initial positions, and with the 0.005 scaling applied) from each of the frames divided by the scaling factor of 0.005, and stores it in an array. In this way, the results of the interpolation carried out by Blender between two keyframes can be acquired. These results are stored in .mat files (e.g., with the scipy.io.savemat function) to be compared with the results of the numerical integration. The error between the numerically integrated results and the position of the bubbles acquired from Blender is shown in Figure 4.

The error towards the end of the simulation occurs because Blender assumes that no further motion takes place after the final keyframe. In order to prevent the object from abruptly stopping at the end of an animation, Blender interpolates the object property, in this case, location, to a smooth stop. This results in the object property having a time derivative that approaches 0 near the end of the simulation. This is observed in Figure 5, where the bubble positions from integration and interpolation between the last two keyframes are plotted. The red line in the plot, which represents the interpolation carried out by Blender, is observed to have a slope of zero towards the 120 s mark, while the blue line, which represents the numerically integrated result, does not, in accordance with the equations of motion. Despite these errors at the end of the simulation, it can be seen that overall, the number of keyframes used in the simulation enabled the results with interpolation from Blender to follow sufficiently well compared to the results from performing the numerical integration. Because the error is greatest toward the beginning and end of the simulation due to the manner in which Blender's interpolation works, it may be useful when performing a physics simulation in Blender to simulate the process in Blender beyond the end time of the simulation to avoid the effect observed in Figures 4 and 5, in which Blender assumes that the object stops moving at the end of the simulation time when that is not implied by the numerical integration result. This also explains a similar error being observed near the start of the simulation, where the objects begin moving from rest. This allows for the results of the animation to replicate the physics more accurately and aids in drawing useful conclusions about the process from the simulation.

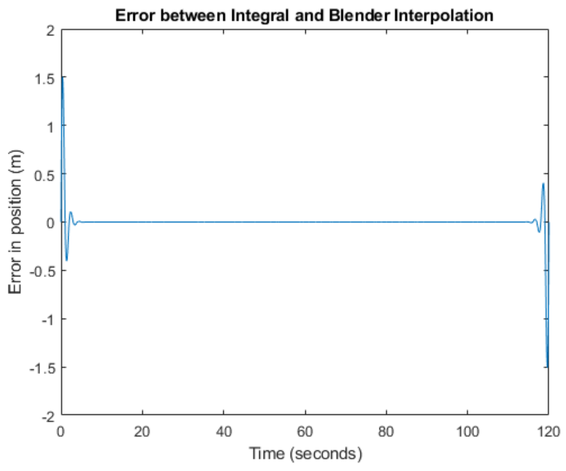


Figure 4. Frame by frame error in bubble position.

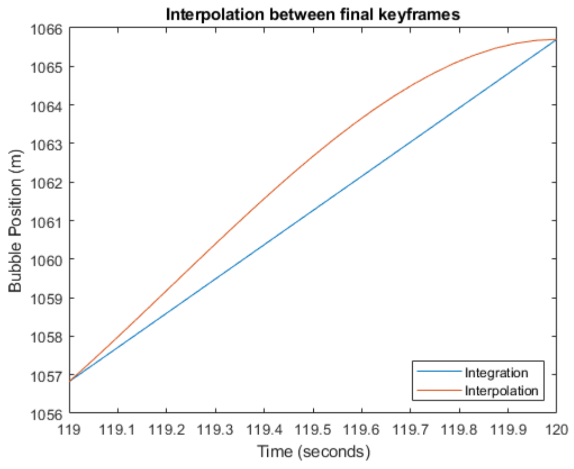


Figure 5. Comparison of numerically integrated and interpolated result for the last second of simulation.

3.1.4. Developing Useful Visuals in Blender

The bubble simulation is considered to be a precursor toward more advanced image-based control studies that could be undertaken, like a full simulation of a flotation process. In this section, we provide several notes on aspects of how to move toward a visually-realistic simulation of froth flotation.

A flotation process involves air introduced into a slurry, which is a mixture of water and treated ore particles [46]. These ore particles are formed by grinding down larger metal ores and treating them with a hydrophobic reagent. The introduction of air agitates the slurry, resulting in the formation of air bubbles. From there, the bubbles rise and come into contact with the hydrophobic ore particles, where they become attached to the bubble surface and rise. This results in the formation of a froth, which is then skimmed off the surface of the slurry and treated to acquire the desired metal from the ore particle. From treating ore particles to reduce their wettability to stabilizing the froth so that it can be skimmed without losing the fine ore particles, every step of the froth flotation process must be carried out carefully. The froth produced in froth flotation cells needs to be chemically analyzed in order to determine which combination of reagent concentration, particle size, and other factors, extract the most ore. However, image sensors can be calibrated to determine these characteristics by correlating representative images against the known results.

Thus, an important characteristic of the development of image-based controllers for a process like froth flotation should be creating models with an appearance that is useful for assessing the performance of an image-based controller through its resemblance to the actual process. The development of a realistic-looking model in Blender requires decisions on the modeling, texture/color, and lighting of a scene. Figure 6, for example, shows a model of foam that might be compared against actual foam from a flotation process to analyze whether it captures key visual characteristics (the set-up of the materials, which controls the "look" of the froth, is shown in Figure 7). In addition, animation in Blender can also be used to capture other physical features of a process, such as the tank shape and also constant movement at a plant, such as an impeller rotating. The ability to modify material properties and lighting in Blender demonstrates its ability to simulate the images captured by a sensor, enabling useful information to be gathered before the control set up is installed in the target process.

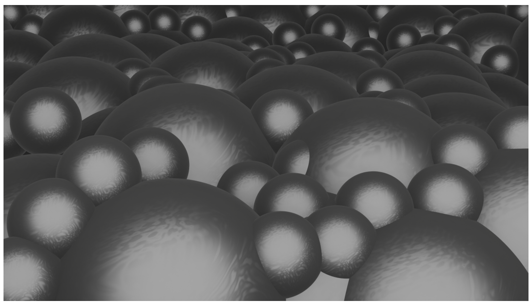
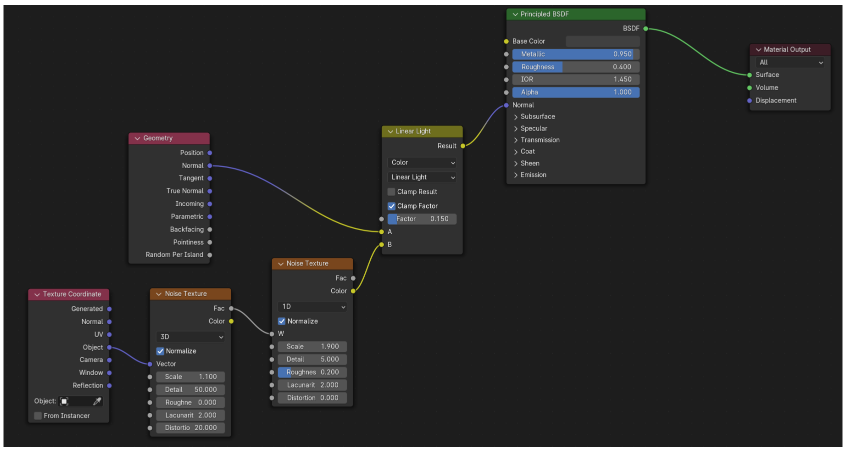


Figure 6. Rendered foam image from Blender.



Texture Coordinate: Object: ; From Instancer ☐

Noise Texture: 3D; Normalize ✓; Scale 1.100; Detail 50.000; Roughne 0.000; Lacunarit 2.000; Distortio 20.000

Noise Texture: 1D; Normalize ✓; Scale 1.900; Detail 5.000; Roughnes 0.200; Lacunarit 2.000; Distortion 0.000

Linear Light: Color; Linear Light; Clamp Result ☐; Clamp Factor ✓; Factor 0.150

Principled BSDF: Metallic 0.950; Roughness 0.400; IOR 1.450; Alpha 1.000

Material Output: All

Figure 7. Material property set up for Figure 6. This is derived from an online tutorial hosted on YouTube [47].

Remark 2. *In general, Blender offers potential to analyze various aspects of image-based control. For example, one might consider multivariate image analysis, wavelet texture analysis, and the gray-level co-occurrence matrix as different image processing methods (or principal component analysis (PCA)) and wish to compare them, specifically for froth flotation applications [40]. Multivariate image analysis (MIA) is a methodology for analyzing multivariate images where the images are analyzed in the form of matrices. MIA can be used to extract features related to the color information in an image by first decomposing the image with the PCA technique, in order to acquire spectral (color-based) features that can be correlated to characteristics of the froth. This is carried out with multivariate image regression (MIR). However, textural features of froth images are better analyzed with gray level co-occurrence matrix and wavelet texture analysis methods. A gray level co-occurrence matrix (GLCM) is a matrix with a number of rows and a number of columns equal to the number of gray levels in the image. The GLCM matrix is then used to extract textural features from the image using statistical methods, such as the variance, correlation, and entropy. It would be expected that Blender may be useful for analyzing not only camera and light positioning but also for comparing different image analysis methods that could be considered. Furthermore, more advanced simulations might be developed by attempting to better correlate images with the physics of the situation. For example, Do [48] describes physics of turbulent flotation, and Cleary et al. [45] describes a detailed graphics modeling strategy for bubbles in liquid. Through more detailed modeling strategies in the future, it may be possible to utilize Blender more for analyzing and modeling image-based control processes in greater detail. However, it should be noted that in general, a traditional process model may not directly translate to an ability to model the process in Blender.*

3.2. Evaluating Identification Techniques for a Controlled System under Image-Based Control in Blender

The prior section described the use of Blender in the modeling of a process that could be placed under image-based control and some of the considerations for checking whether the modeling efforts are sufficient to represent a system. In this section, we focus on a process under image-based control in Blender and discuss how Blender helps to reveal characteristics required by either the image processing algorithm or process design to enable the image-based control to be effectively carried out. The process under stochastic control in this section is a nanorod that was used in [39] as a precursor simulation toward considering control of directed self-assembly (though [39] did not perform image-based control manipulations with Blender). The control development in this section follows that in [39], but we present the details of our exact implementation for completeness and replicability of our work. After discussing the simulation setup, we describe the image-based control framework and how it aids with understanding some of the potential pitfalls of an image-based controller for the process, as well as ideas for overcoming them.

Directed self-assembly (DSA), the motivation behind simulating the optical control of the nanorod, is a manufacturing technique that has been gaining research interest in recent years. The technique has its roots in self-assembly, where instead of external control actions, molecules are assembled by modifying them such that the desired configuration can be acquired stably, or by similarly modifying a surface on which the self-assembly takes place, commonly referred to as a substrate. This idea has been explored in the field of biology specifically by using the DNA [49,50] and in the polymer sciences through experiments on block copolymerization (BCP) [51,52]. The modification of a base substrate to direct multiple species to bind and form a desired configuration is generally seen as the first attempt at directing the self-assembly of particles, though this idea overlaps with those of regular self-assembly. The improvement of BCP for directed self-assembly demonstrates the capability of this method, by forming highly precise nanostructures on a pretreated substrate [53,54] or using forces of the medium to achieve similar results [55]. External forces can also be used in the absence of substrates, sometimes to improve existing self-assembly [56] or to drive particles into positions to facilitate further interaction [57]. The ability to use image-based control in self-assembly applications has been explored in the work of Tang et al. [58]. In this work, images are processed and used to form the feedback of a control loop, where electric fields are used to manipulate the free energy landscape and ensure the colloidal particles form the desired morphology. In that example, the arrangement of the individual particles in the medium, the stability of their configuration, and speed of assembly can be observed directly through imaging techniques. A virtual model of these systems is what an IBC testbed would hope to emulate, simulating ideal conditions for optimum self-assembly, as well as an image-based observation of the process.

Motivated by the potential utility of image-based control for self-assembly, as demonstrated in this prior work, this section focuses on the self-assembly precursor simulation from [39], as discussed above, of a nanorod undergoing Brownian motion in a two-dimensional space. Stochastic control is used to carry out navigation of the rod to the center of the control space. Image-based control is simulated in Blender with the virtual camera that captures images of the rod, which are then used to determine the state of the rod and carry out the optimal action. The optimal policy (consisting of on/off-type control actions) is acquired by performing stochastic optimization, as detailed in Section 2.2.

3.2.1. Equations of Motion for a Self-Propelled Rod

Since the rod simulation is used in the PhD dissertation by Yang [39] as a precursor to self-assembly studies, the rod's motion in free space is modeled using 2D Brownian motion, according to Equations (9) and (10):

$$\begin{aligned}\mathbf{r}(t+\Delta t) = \;& \mathbf{r}(t) + \tfrac{\mathbf{D}_t}{kT}\cdot\mathbf{F}\Delta t + \Delta\mathbf{r}^B \\ & + v\cos(\phi)\Delta t\,\mathbf{e}_1 + v\sin(\phi)\Delta t\,\mathbf{e}_2 \\ \phi(t+\Delta t) = \;& \phi(t) + \tfrac{D_r}{kT}\mathbf{T}\cdot\mathbf{e}_3\Delta t + \Delta\phi^B \\ \mathbf{D}_t = \;& \begin{pmatrix} D_{xx} & D_{xy} \\ D_{yx} & D_{yy}\end{pmatrix}\end{aligned} \tag{9}$$

$$\begin{aligned}\langle\Delta\mathbf{r}^B\rangle = 0 \quad & \langle\Delta\mathbf{r}^B, (\Delta\mathbf{r}^B)^\intercal\rangle = 2\mathbf{D}_t\Delta t \\ \langle\Delta\phi^B\rangle = 0 \quad & \langle\Delta\phi^B, \Delta\phi^B\rangle = 2D_r\Delta t\end{aligned} \tag{10}$$

In these equations, $\Delta\mathbf{r}^B$ and $\Delta\phi^B$ represent the changes to the 2-dimensional position and the orientation of the rod (i.e., the angle between the line along the axis of the cylinder and the positive x axis) under Brownian motion and are normally distributed random variables. $\mathbf{D}_t$ (with components D_{xx}, D_{xy}, D_{yx}, and D_{yy}) and D_r represent translational and rotational "diffusivities", which characterize the random motion of the rod due to particles of the medium. The velocity of the rod under the control action is represented by v and is only non-zero when the control action is on. Since no force or torque fields act on the rod (hence, we refer to it as "self-propelled"), the terms $\mathbf{F}$ and $\mathbf{T}$ are set to 0. Vectors $\mathbf{e}_1$ and $\mathbf{e}_2$ represent unit vectors along each of the two dimensions, and $\mathbf{e}_3$ represents the unit vector normal to the 2D plane. The temperature of the system is represented by T, and k represents the Boltzmann constant. From the definition of the translational diffusivity tensor $\mathbf{D}_t$, the components of $\Delta\mathbf{r}^B$ are coupled. For a cylinder, $\mathbf{D}_t$ is parameterized with $D_\parallel$ and $D_\perp$ [59], diffusivities along and perpendicular to the cylinder axis, respectively. The following equation (Equation (11)) simplifies the tensor to a single parameter by assuming $D_\parallel \approx D_\perp$ as follows:

$$\begin{aligned}& \mathbf{D}_t = \mathbf{n}\mathbf{n}^\intercal D_\parallel + (\mathbf{I} - \mathbf{n}\mathbf{n}^\intercal)D_\perp \\ & \text{where } \mathbf{n} = \begin{pmatrix}\cos(\phi)\\ \sin(\phi)\end{pmatrix} \\ & \text{If } D_\parallel \approx D_\perp \\ & \mathbf{D}_t = \mathbf{I}\cdot D_\perp \\ & \quad = \begin{pmatrix} D_\perp & 0 \\ 0 & D_\perp\end{pmatrix} \\ & \implies D_{xx} = D_{yy} = D_\perp = \bar{D}_t\end{aligned} \tag{11}$$

where $\mathbf{I}$ is the identity matrix.

This decouples the movement in the orthogonal directions, as well as transition probabilities for location and orientation. The equation of motion used for the simulation then simplifies to Equation (12) as follows, where the velocity v is either zero or a constant, depending on the choice of the control action, i.e., on or off:

$$\begin{aligned} x(t+\Delta t) &= x(t) + \Delta x^B + v\cos\phi\Delta t \\ y(t+\Delta t) &= y(t) + \Delta y^B + v\sin\phi\Delta t \\ \phi(t+\Delta t) &= \phi(t) + \Delta\phi^B \\ \langle \Delta x^B \rangle = 0 \quad & \langle \Delta x^B, \Delta x^B \rangle = 2\bar{D}_t\Delta t \\ \langle \Delta y^B \rangle = 0 \quad & \langle \Delta y^B, \Delta y^B \rangle = 2\bar{D}_t\Delta t \\ \langle \Delta\phi^B \rangle = 0 \quad & \langle \Delta\phi^B, \Delta\phi^B \rangle = 2D_r\Delta t \end{aligned} \tag{12}$$

3.2.2. State Space and Transition Probabilities

The optimal policy generation method from Section 2.2, which we wish to employ to control the rod, uses a discrete state-space. In general, the space in which a rod moves is a continuous space; it is, therefore, necessary to determine how to translate the continuous state space into a discrete state space for the purpose of control. To do this, we select a portion of the free space in an area 101 μm × 101 μm around the target and set it as the neighborhood around which the policy is generated. To create the discretized state space, this space is first discretized into 1 μm × 1 μm zones, and the rod's position can take values at the center of each square. The orientation is discretized as sectors of $\pi/4$ radians each, centered on each of the cardinal and subcardinal directions, as in Figure 8. The set of all possible discrete states can then be defined as a three-dimensional array, where the first two indices represent the position of the nanorod in the space, and the third index represents the orientation sector that it occupies. This set is defined in the following equation (Equation (13)), where the intersection with $\mathbb{Z}^+$ indicates that x, y, and ϕ are integers within the specified intervals:

$$S = \{(x, y, \phi) : x, y \in [0, 100] \cap \mathbb{Z}^+, \phi \in [0, 7] \cap \mathbb{Z}^+\} \tag{13}$$

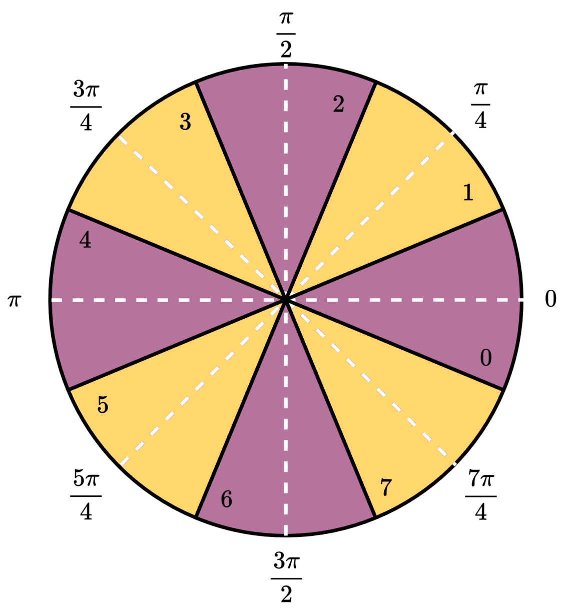


Figure 8. Discretization of orientation into 8 sectors. The numbers in each sector denote its index. The dashed white lines indicate the central angles for each sector, the purple sectors are centered on the cardinal directions, and the yellow ones are centered on the sub-cardinal directions.

The dynamics of the rod as presented in Equation (12) are presented in discrete time and are used to generate the transition probabilities associated with each state. The transition probabilities map the probability associated from one state to another for each action, i.e., $P : A \times S \times S \mapsto (0, 1)$, and based on the discretization of the state space in Equation (13), the total number of transition probabilities would be $2 \times (101 \times 101 \times 8)^2$. The normal distributions of Δx^B and Δy^B imply that the probability of moving to a new position tends towards zero outside of a small neighborhood around an initial position. Hence, a neighborhood of a 19 μm × 19 μm grid around the initial position of the rod is set as the maximum possible displacement in a single time step. The transition probabilities are then acquired through Monté Carlo trials. Monté Carlo trials are particularly useful in extracting equivalent discrete probabilities from continuous random variables. First, a large number of values for the random variables (trials) are generated, according to their known distributions. Then, the probability associated with each outcome is found by counting the number of times the outcome is observed and dividing it by the total number of trials. Three sets of random data are generated: the position Δx^B and Δy^B and the orientation $\Delta\phi^B$. Table 2 details the parameters in Equation (12) used to generate the trials. The probability associated with how far the nanorod travels after one time step is termed as positional probability. This is calculated by first dividing the neighborhood of a 19 μm × 19 μm grid around the initial position of the rod into 1 μm × 1 μm zones and determining which zone the nanorod ends up in for each trial (based on the center of the zones). Dividing the number of times the nanorod ends up in each zone by the total number of trials yields the positional probability. This positional probability is then multiplied by the discrete probabilities associated with the orientation. Similar to how the positional probability is acquired, orientation probabilities are acquired by counting the number of times the nanorod ends up in each orientation sector, as described in Figure 8, given an initial orientation sector. This operation is repeated for each control action, where additional components of the velocity $v \cos\phi\Delta t$ and $v \sin\phi\Delta t$ are added to Δx^B and Δy^B, respectively, for the on control action, as described in Equation (12). Hence, the total number of transition probabilities reduces to $2 \times 8 \times 8 \times 19 \times 19$ for a given initial position, where the first index represents the chosen control action, the second and third indicate the initial and final orientation of the rod, and the fourth and fifth indicate the final position of the nanorod relative to the center of the neighborhood. These transition probabilities are stored in a matrix P. The transition probability associated with the rod, starting in orientation sector 0, moving 8.5–9.5 μm towards the positive end of the x-axis, with a final orientation in orientation sector 0, and moving under Brownian motion alone (i.e., when the chosen control action is off), would be stored at $P(0, 0, 0, 9, 18)$. Figure 9 displays the transition probabilities for the rod with orientation sector 0, associated with final positions for both on and off control. Each square represents the probability of the rod acquiring that position after Δt time has passed, given its initial position at the center of the heatmap. The colorbar represents the values of the probability of the nanorod occupying a specific position once it starts at the origin. Figure 10 displays how changing orientation affects the transition probability. Since the change in orientation is also normally distributed around 0, the observed transition probabilities when orientations change (compared to the initial orientation) is lower than those where the orientation remains the same as the initial orientation.

Table 2. Probability Generation Parameters.

Parameter	**Value**
Translational diffusivity, $\bar{D}_t$	0.5 μm^2/s
Rotational diffusivity, D_r	0.548 rad^2/s
Sample time, Δt	1 s
Discretization length, $\Delta x = \Delta y$	1 μm
Velocity of rod under control, v	2.5 μm/s

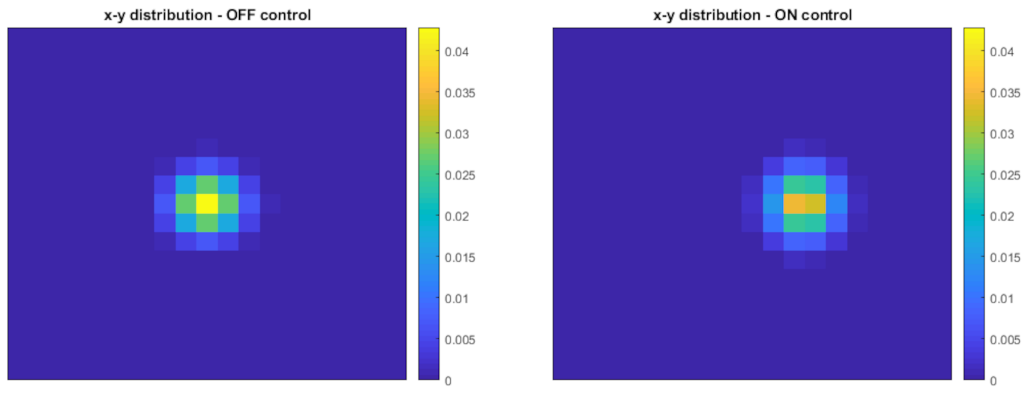


Figure 9. Positional probability distribution for the nanorod in sector 0 of orientation. The figure on the left shows the probability distribution when the control action is off. In this case, the rod starts at the center of the figure and has the highest probability to stay at the center, with lower probabilities moving away from the center. The figure on the right shows the probability distribution when the control action is on. In this case, the rod starts at the center of the figure and has the highest probability to move slightly to the right of the center, with lower probabilities moving away from the right-shifted region of highest probability.

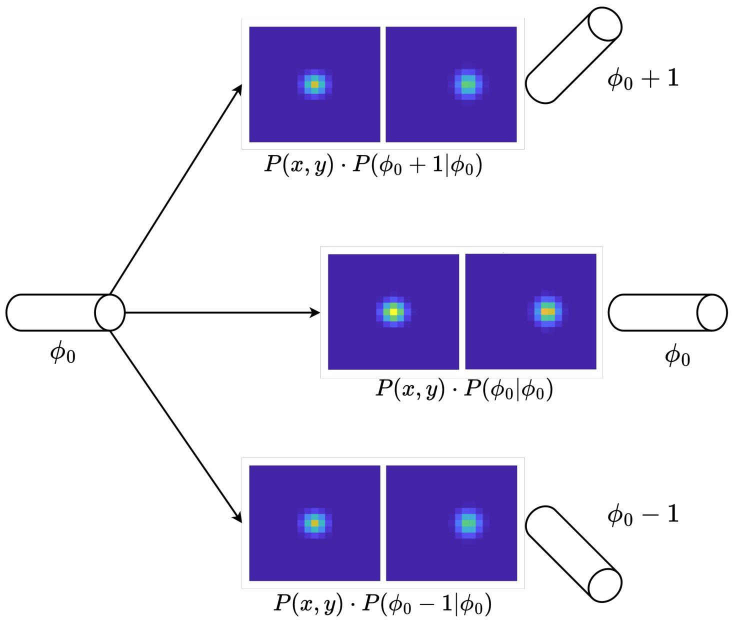

$\phi_0 + 1$
$P(x,y) \cdot P(\phi_0 + 1|\phi_0)$
ϕ_0
$P(x,y) \cdot P(\phi_0|\phi_0)$
ϕ_0
$\phi_0 - 1$
$P(x,y) \cdot P(\phi_0 - 1|\phi_0)$

Figure 10. Transition probability distributions for 3 adjacent orientations. The cylinders represent the orientations of the nanorod, with the one on the left representing the initial orientation and the three on the right representing three possible final orientations. The associated transition probability heatmaps show that since the rod is most likely to keep its orientation, the heatmaps above and below the center line show lower probabilities when compared to the center. The colorbar used to indicate the relative magnitude of the transition probabilities shares the same scale and is identical to that in Figure 9. $P(x,y)$ indicates the probability of a certain (x,y) location in space given an initial state at the center, and $P(\phi_j|\phi_0)$, where ϕ_j is either $\phi_0 + 1$, ϕ_0, or $\phi_0 - 1$, indicates the probability of achieving certain orientations given an initial orientation of ϕ_0.

These reduced probability matrices (i.e., considering probabilities within only a fraction of the total state space) are then translated to the corresponding window in order to carry out the value iteration algorithm, as shown in Figure 11. With the state space discretized and the transition probabilities calculated, the policy can now be generated in accordance with Bellman's optimality condition.

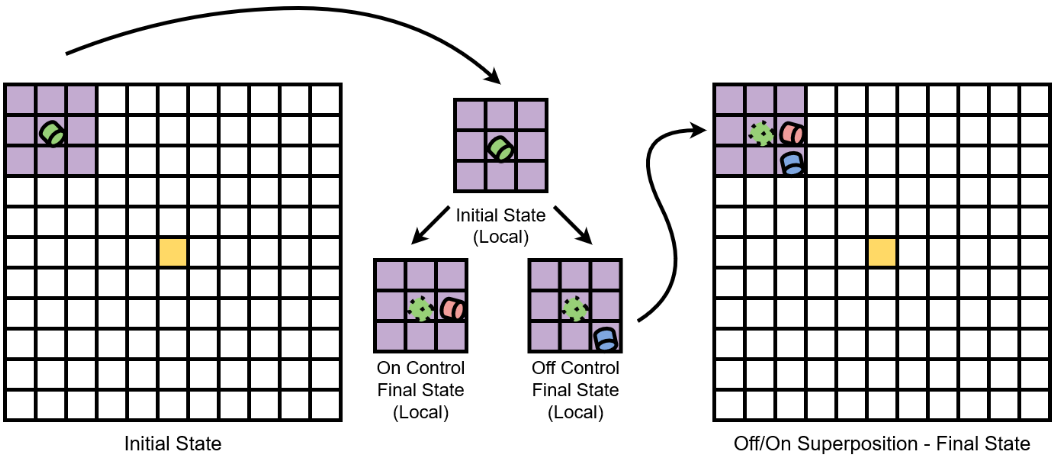


Figure 11. Using reduced transition probabilities to calculate overall transition probabilities. The left figure starts with defining where the neighborhood of possible positions is (indicated in purple), centered on the initial position of the nanorod, depicted as a green cylinder. The middle figures demonstrate how transition probabilities are used to determine possible states, where the red cylinder shows a potential final state under the on control action, and the blue cylinder under the off control action. Furthermore, the figure on the right shows how the reduced transition probabilities relate to the transition probabilities of the entire state space.

3.2.3. Policy Generation

From Equation (2), the optimal value function is described as a weighted sum of the value functions of possible future states and the cost function at the current state. Generally speaking, the cost function is a representation of the cost to maintain a certain state. However, since the final state (considered to be navigating the rod to the center of the portion of free space being simulated) is known in this simulation, the cost function is designed such that its value at the center of the free space is the minimum. From Equation (3), the value of every state beyond the boundary of the 101 μm × 101 μm region must be the sum of its cost as well as the value of neighboring states. From the discussion in Section 3.2.2, the set of possible future states is restricted to a small neighborhood of states surrounding the initial position. If a portion of the future states is outside the boundary of the modeled space, and since states outside the boundary cannot be iterated, it is necessary to have a placeholder value function instead, as shown in Figure 12.

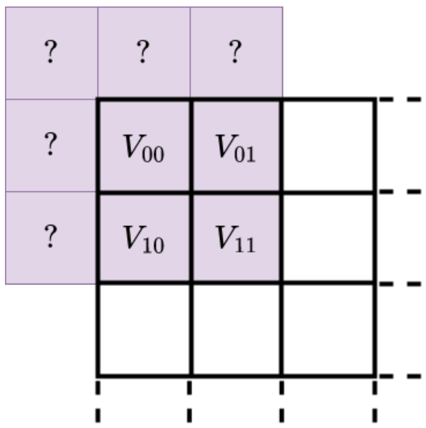


Figure 12. Edge effects in 2D presentation. When determining the neighborhood of possible positions (purple) around the initial nanorod position, if the rod lies close enough to the boundary, the neighborhood can include positions that lie outside the boundary (i.e., where the question marks are located in the figure). The values V_{pq}, $pq = 00, 01, 10$, and 11 indicate the value function values at different positions.

Since the value function is a function of cost, and the optimization procedure involves the minimization of cost, we can assume that states outside the boundary have higher value function mappings than those within. For states with locations infinitely far away from the target, the result of the value function should be maximum and approximately equal to the value of the neighboring states. In order to meet this requirement, and following the description of the state space from Equation (13), the cost function is described in Equation (14):

$$C(s) = \begin{cases} 0 & \text{If } s \notin S_{\text{target}} \\ -1 & \text{If } s \in S_{\text{target}} \end{cases} \tag{14}$$

$$S_{\text{target}} = \{(x, y, \phi) : x, y = (50, 50), (x, y, \phi) \in S\} \tag{15}$$

S_{target} is chosen such that the target area lies at the center of the modeled space. Since the value function is recursively defined as linear combinations of the cost function, it is bounded by the maximum of 0. This is selected as the placeholder value function result outside the boundary of the simulation, as displayed in Figure 12. With this feature in place, the Bellman update rule (Equation (3)) is carried out for every state. Figure 13 displays the converged policy with $\gamma = 0.8$ (iterations stopped if the sum of the value function results for every state between two consecutive iterations was below a threshold of 10^{-5}).

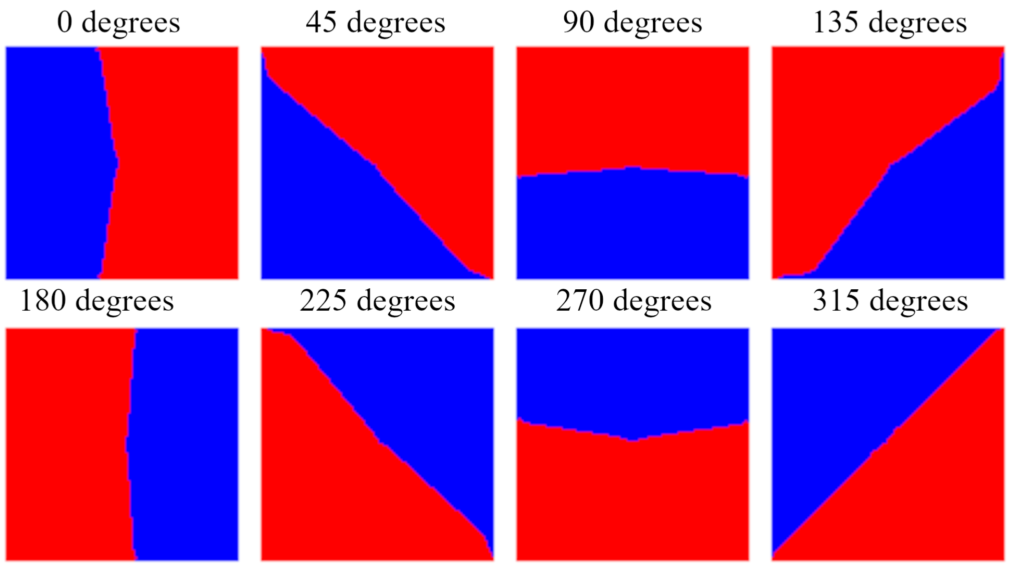


Figure 13. Optimal control policy based on location and orientation with $\gamma = 0.8$. Blue regions are regions where the control action is on; red regions are regions where the control action is off.

3.2.4. Blender Implementation of Rod Dynamics

Blender is an open-source computer graphics program with capabilities in modeling, animation, and rendering, along with a Python programming interface, which facilitates visualization of the control policy described in Section 2.2. The nanorod is modeled as a cylinder (given initial values in Blender with scale = [0.02,0.02,0.10], rotation_euler = [0,$\frac{\pi}{2}$,0], and location = [$-$0.5,0.5,0.01], and a seed for random number generation of 252), and its motion is restricted to the x-y plane, as in Figure 14. As discussed in Section 3.2.1, the control action moves the rod along its axis at a fixed velocity.

Because both the position and orientation of the rod are important for selecting the control action to be applied in this study, a method for distinguishing the rod's orientation from the image must be obtained. A challenge in this direction is that the rod is a cylinder and, thus, symmetric, unless distinguishing marks are placed on it. To enable the orientation to be ascertained, the direction of the rod is determined by color coding the "front" end of the rod white and the "back" end black. The body of the rod is colored blue, and a contrasting backplate is modeled as a red plane. These various modifications indicate several learnings from the Blender simulation that would aid with designing an image-based controller for an actual process. First, they elucidate that without the color coding, it may be difficult to accurately determine the rod's orientation without a more complex image processing strategy (e.g., a type of edge detection). This helps to indicate some of the options available to a process or control designer for attempting to set up such a system successfully in a real plant before the time to build it. Furthermore, the need to ensure that there is sufficient contrast between the background and the color of the rod (and its ends) is also highlighted, again showcasing some of the features of the system design that are important for achieving the desired control goals.

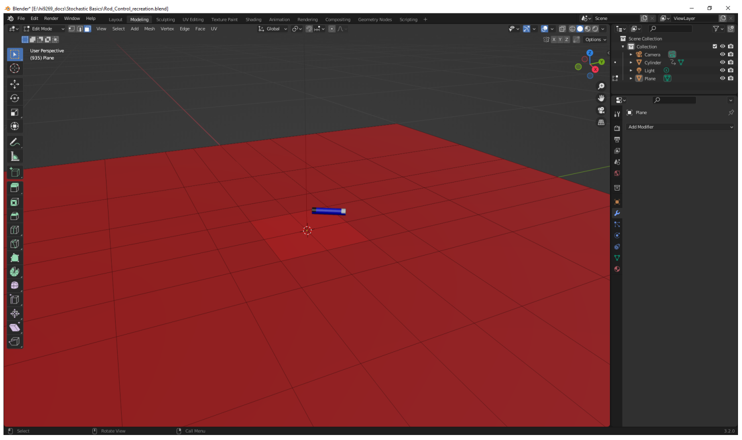


Figure 14. Models of rod and contrasting backplate in Blender.

As discussed in Section 3.2.1, Brownian motion is simulated as a random walk in discrete time, and the motion is characterized in continuous time as the Weiner process [60]. The distance unit of the parameters used to describe the Brownian motion of the rod and the control action is scaled up, such that 1 m in the Blender model corresponds to 50.5 µm. The simulation is run at 24 frames per simulation second, with the position and orientation updated for each frame. The updates are carried out according to Equation (12), with $\Delta t = 1/24$ s of simulation time. With the rod dynamics now in place, Blender's image capture technology can now be explained.

3.2.5. Image Capture and State Measurement

Three-dimensional animation software suites generally employ a virtual camera to render animations with desired position and lighting. In order to capture the state accurately, the camera in Blender is positioned above the contrasting backplate with its line of sight perpendicular to the plane. An area light source is set above the backplate to illuminate the control area and the nanorod uniformly. Each end of the nanorod being colored differently allows the detection of the position and orientation of the rod. In the generation of the optimal policy, the rod is assumed to be a point object with respect to the discretization of the location. The choice of black and white ends is due to the representation of color in the image capture. Digital image capture uses the additive color primaries: red, green, and blue. White pixels have the highest combination of these primaries, and black pixels have the lowest. In order to minimize the possible effects due to lighting conditions, sums of the color channels around points of interest are used to determine color. This is done by determining the sum of the RGB channels for all pixels in a 3×3 window centered on the pixel being inspected, and taking the sum of them to determine the color (i.e., the window with the smallest sum is considered to identify the black end and that with the largest sum is considered to identify the white end). This also informs our choices for the rod and contrasting backplates, to prevent the body of the rod and the medium from being detected by the image capture algorithm. The locations of the ends of the nanorod are used to determine the discretized location by averaging the coordinates of each end to get an approximate location of the mid-point of the rod. Additionally, the tangent inverse of the negative of the slope of the line through each end is used to determine the orientation. Since digital array indices are numbered from the top left corner, i.e., the index (0, 0) is at the top left corner, and the row index corresponding to the y coordinate increases in the downward direction, the tangent inverse of the negative slope is used to obtain the orientation with respect to having the zero orientation at the standard location of zero degrees in polar coordinates (i.e., on the right). Figure 15 details the operations that take place at each image capture. In general, an image-based control testbed can help with analyzing effects such as lighting effects in advance of attempting to implement and design an area instrumented with image-based control.

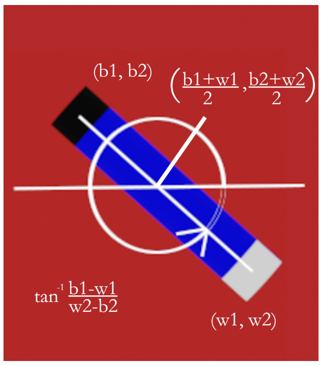

(b1, b2)
$\left(\frac{b1+w1}{2}, \frac{b2+w2}{2}\right)$
$\tan^{-1}\frac{b1-w1}{w2-b2}$
(w1, w2)

Figure 15. State determination from image information. (b_1, b_2) signifies the array index of the black end of the rod from the upper left, and (w_1, w_2) signifies the array index of the white end. The average of these points is listed and the argument of the inverse tangent, used in the computations of the orientation, is the slope between the points in the Cartesian coordinate system.

These images are generated with render functions that come built-in with the Blender Python library. The policies generated as per Section 3.2.3 have a much lower discretization than the resolution of the image returned in the image capture algorithm. The images are of the size 1920 $\times$ 1920, while the location component of the state space, as described in Equation (13), is of the size 101 $\times$ 101. These images are compressed in order to match the scale of discretization of the policy. Specifically, the number of pixels that are used to represent the image is changed because the image captured by Blender has a resolution of 1920 $\times$ 1920, and in order to identify what the discretized location of the rod is, it is necessary to make this size match that of the policy (i.e., 101 $\times$ 101). The state determination and image compression are carried out using the `resize` function in Pillow. At every state observation (which takes place at every sample time), an image is explicitly rendered from the current frame and stored locally. This image is then imported into the running Python script with Pillow, compressed, and processed for state determination. The position and orientation of the rod updated with Brownian motion dynamics are compared with the values acquired from state determination in Table 3. Specifically, the L^1 norm of the error between the actual position and the position obtained by the image processing algorithm is reported in Table 3 (and is calculated as the sum of the absolute values of the differences between the first position index and the second position index between the actual values (“Position Blender” in the table) and the values obtained from the image processing algorithm (“Position Image” in the table)). As shown in the fourth column of Table 3, the maximum L^1 norm error computed in this way is 2, indicating that the position values for both position indices, as obtained from the image processing method, were close to the correct values. In addition, the absolute error between the sector identified for the orientation using the image processing (“Orientation Image” in the table) and the actual sector (“Orientation Blender” in the table) is reported in Table 3. In the full data set for the 1200 s simulation, there are cases where an error was recorded in the sector determination from the images when the rod was actually in Sector 7 (since there are 8 sectors, this can also be considered to be the sector -1) but it was stated by the image processing strategy to be in Sector 0. The absolute error (considered to be the absolute value between the difference between the 0 and -1 sector values) in that case is 1, suggesting once again that the image processing algorithm performed well in many cases in determining the orientation of the rod correctly. The results reflect that the image analysis often produced correct orientations and that the position error was relatively low, suggesting that this control policy may perform well but that there is also potential for improvement of, for example, the position sensing if desired. State determination from the rendered image is one of two core functions of the Python script. The other is rendering the animation, and it is discussed in the following section.

Table 3. Deviation and error of state determination for selected frames of a 1200 s simulation.

Time Seconds	Position Blender	Position Image	L1 Norm Error	Orientation Blender (Sector)	Orientation Image (Sector)	Absolute Error
80	(28, 30)	(29, 31)	2	0	0	0
160	(32, 32)	(33, 33)	2	4	4	0
240	(32, 32)	(32, 33)	1	0	0	0
320	(32, 34)	(32, 34)	0	1	1	0
400	(37, 37)	(38, 38)	2	3	3	0
480	(36, 36)	(37, 37)	2	3	3	0
560	(36, 38)	(36, 38)	0	7	7	0
640	(45, 44)	(45, 44)	0	0	0	0
720	(48, 49)	(49, 49)	1	5	5	0
800	(50, 48)	(50, 47)	1	4	4	0

3.2.6. Animation and Control Execution

In Section 3.2.4, it is explained that the position and orientation are updated for every frame of the animation. According to Section 2.3.2, since keyframes update object properties in the simulation, they are used to update the position and orientation of the rod in the animation. In this work, however, the keyframes help simulate the interplay between the smaller timescales of the rod dynamics and the image-based control. Specifically, the sampling time for the controller as listed in Table 2 is 1 s, which is also when each keyframe is inserted. The Brownian motion deviation variables are updated every 1/24 s in the Python programming interface, as detailed in Section 3.2.4. Since the deviation variables $\Delta\mathbf{r}^B$ and $\Delta\phi^B$ are generated with different time scales than what is used to determine the policy, the simulation demonstrates sample-and-hold control of a Brownian motion process. A keyframe is inserted at every sampling time in this experiment, during which the image is captured, the state is identified, and the control action is determined. The control is actuated in a sample-and-hold fashion, keeping the velocity constant regardless of changes in state until the next sampling time occurs. Figure 16 is a compilation of locally stored images used for the state determination, which are also from keyframes.

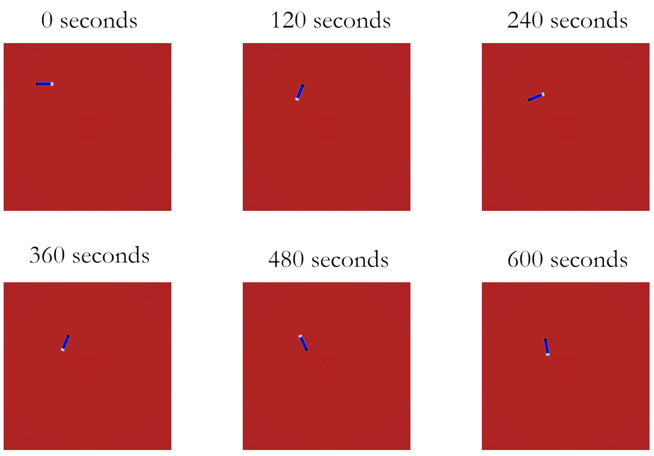


Figure 16. Rendered keyframe images for state determination at different simulation times.

The Blender script is run for 1200 s of total animation time; however, due to the non-determinism of the system, the agent can enter the target space within any amount of time that, on average, scales with the distance of the initial position to the target for a given initial orientation. This series of frames indicates that though the rod starts off the target and takes some time in moving toward it under its stochastic dynamics, it eventually reaches the target, suggesting success of the image-based control strategy for this example.

3.2.7. Opportunities for Image-Based Control Testing in Blender Based on the Stochastic Nanorod

The sections above demonstrate that Blender is successfully used to replicate Brownian motion dynamics of the nanorod in 2-dimensional space, and the optimal stochastic policy is implemented using a virtual image-based controller. By updating the position and orientation of the rod every frame ($\Delta t = 1/24$ s) and picking an optimal action at every sampling time (where keyframes are also inserted), the simulated process attempts to show how the sample-and-hold control policy performs on a process that is updated on a smaller timescale. One aspect of setting keyframes for the simulation of the rod, compared with the bubble of Section 3.1, is that the bubble was taking a straight path at a constant velocity such that, except near the beginning and end of the simulation where the starting and stopping behavior of the bubble in Blender created a mismatch between the numerical integration result and the Blender simulation result, no error was observed between the interpolated position values (between keyframes) by Blender and the values obtained from numerical integration. However, in this simulation involving the rod, due to the stochastic behavior, we would not expect that the results with applying keyframes every frame versus every several frames would cause the position and orientation of the rod in Blender to be the same in both cases. This indicates that the frequency with which keyframes are set can cause different accuracy for different dynamic systems.

The color information is summed within a 3×3 window to detect the rod and extract information about its state features. Image capture by the software and additional image processing with the Python Pillow library is demonstrated, and the errors between the states of the system returned from the virtual environment and those extracted from the image are compared.

Since Blender can render animations, it provides the ability to visually demonstrate and inspect the execution of an image-based control algorithm. When attempting to visualize the nanorod approaching its destination using a rendered video of the animation, the target can be highlighted in the video. In such a case, a section of the contrasting backplate around the target could be colored differently, for example, when a video of the simulation is rendered to aid in showing that the rod approached its target with time. However, care must be executed when attempting to improve visualization of aspects of the simulation such as this to ensure that the visualization aids do not interfere with the behavior of the image-based control algorithm. For example, attempts were made to aid with visualizing the rod moving toward the target by making the target area visible (i.e., a different color than the rest of the backplate) during the execution of the image-based control policy. However, this resulted in failure of the color detection strategy, in part due to the summing of the color channel values in a small neighborhood around the nanorod, which would include portions of the contrasting backplate as the rod moved close to the target. In the simulations performed, a yellow colored target section registered as the white end of the nanorod, since windows of color summing would place the sum total of the color channels in the target section higher than the actual front end of the nanorod due to the backplate being included in windows around the front. A darker color than the backplate would result in similar misidentification of the black end of the nanorod. These issues might be addressed with the use of image convolutions, such as edge detection, in order to further improve state identification. Alternatively, Blender allows for the editing of object properties after keyframes are inserted, which allows the target area to be colored in after the simulation is run, in order to display it in rendered videos. While this allows for highlighting of the simulation features, it is not representative of the images used by the virtual camera for state identification. These studies highlight that making the simulation results more easily understood to a viewer by updating the simulation to include new visualization aspects may affect how true the simulation results remain to what they would be without the additional visualization tools.

Another consideration that could be tested in Blender is cybersecurity or privacy strategies for image-based control systems. For example, one might wish to make the policy between the states and the optimal control actions, as well as the images of the process, unclear to an eavesdropper, even if the policy was to be stored on the Cloud and the images needed to be transferred back and forth between the process and the Cloud for processing for state determination and control action selection. One idea for attempting to obscure, for example, the series of images in Figure 13 is to attempt to utilize a policy inspired by homomorphic encryption being combined with control [61]. In particular, we consider the case that we do not wish to send the actual image to the optimal control look-up strategy, but we wish to process the image locally and then send the state to the Cloud to retrieve the optimal control action from the policy stored there. Since the policy is determined from each state (a 3-dimensional quantity) being mapped to an optimal action, one way of attempting to make the state unclear to an eavesdropper on the Cloud would be to shuffle the full look-up table according to a mapping from the discrete state space to a new value, where that new value would be associated with some control action that could then be returned from the Cloud. This prevents the need to send the actual value of the state to the Cloud, and instead, only the mapped state value is sent. A similar shuffling/mapping idea can also be applied to the optimal action, if it is desired to make the control action returned obscure to an eavesdropper as well. With a strategy for generating the images of the process using Blender, one could simulate this full technique or other cybersecurity and privacy strategies.

The use of Blender opens the possibility of virtually testing various image processing techniques in order to rapidly test novel image-based control algorithms. These various insights regarding image-based control of this process are some of the benefits of utilizing an image-based control testbed like Blender. Additional insights such as camera position, suitable lighting, and minimum resolution for state detection can be simulated in Blender in order to test configurations to test advantages.

3.3. Modeling Sunlight Effects on Outdoor Reactors in Blender

Weather can play a role in the effectiveness of chemical process control policies where equipment is not housed in a controlled environment. The impact of weather on large outdoor reactors can be measured through a variety of methods, but attempting to assess weather changes could aid with reducing plant/model mismatch in model-based control formulations and aid with potentially forming more physics-based models by avoiding treating weather variations as a type of unknown process disturbance affecting a measured state and causing it to deviate from standard operating conditions. In the work of Zavala et al. [62], weather forecasting is used to determine economically optimal actions. With the advent of solar technology, the determination of weather phenomena becomes more relevant, allowing the prediction of energy availability and aiding in ensuring demands can be met by combining a variety of power sources. Going a step further, in the work of Rowe et al. [63], a hybrid solar–electric reactor is designed and then investigated for optimal model predictive control design. The results determined a control setup that accounts for varying sunlight in order to potentially minimize thermal fatigue of the reactor. The ability to simulate the visual recognition of different weather conditions could prove advantageous to determine control parameter adjustments online as weather changes, potentially before its effects show up as measured deviations from normal operation.

An important question in how to set up an advanced disturbance-handling framework, however, that takes advantage of images for identifying different weather patterns and then adjusting control laws as required, is that it is not clear how to adjust the controllers, or whether different visual scenarios might correspond to very different needs in terms of how to update the controller parameters. An image-based control testbed in Blender could help in moving toward addressing some of these questions. However, modeling of weather in Blender can involve, for example, changing lighting model parameters. Lighting models in Blender have non-physical parameters to typically aid in the artistry of lighting for animation. For an engineering testbed application, ideas must be proposed regarding how to make images that correspond to different weather conditions and then translate these into data for a controller as precursors toward more advanced simulation concepts that might be more rigorously compared with more traditional disturbance-handling methods like offset-free model predictive control [64] or online triggering of data-driven model updates [65].

To move toward addressing the potential of Blender for aiding in testing ideas for image-based disturbance-handling methods for model-based control, in this section, a tank reactor that is exposed to outdoor sunlight is simulated in Blender. Using the virtual camera setup in Section 3.2.5, images are captured in order to determine the ambient heat effects on the temperature of the reactor. A model of the tank reactor that takes into account ambient heat due to sunlight is used to simulate the process. The controller uses detected sunlight in order to predict states of the reactor, which are acquired by calibrating the images from the virtual camera with lighting strength in the simulation.

3.3.1. Model Equations for Tank Reactor

The reactor considered in this section is a continuously stirred tank reactor, simulated with a system of ordinary differential equations, which are integrated numerically using the explicit Euler method. The reactor model is detailed in Equation (16), as follows:

$$\begin{aligned}\frac{dC_A}{dt} &= \frac{F}{V}(C_{A0} - C_A) - k_0 C_A^n \exp\left(-\frac{E_A}{RT}\right)\\ \frac{dT}{dt} &= \frac{F}{V}(T_{A0} - T_A) - \frac{k_0 C_A^n (\Delta H_R)}{\rho c_p} \exp\left(-\frac{E_A}{RT}\right) + \frac{Q}{\rho c_p V}\end{aligned} \tag{16}$$

The reactant concentration C_A and the temperature T are the state variables, while the inlet concentration C_{A0} and the heat supply Q are the manipulated variables. The design, reaction, and simulation parameters and their respective values are listed in Table 4.

Table 4. Reaction Parameters.

Parameters	Value
Rate constant k_0	8.46×10^6 $(\mathrm{m^3 kmol^{-1}})^{n-1}\mathrm{h^{-1}}$
Activation energy E_A	5×10^4 kJ/kmol
Density of reaction medium ρ	10^3 kg/m^3
Molar heat of reaction ΔH_R	-1.15×10^4 kJ/kmol
Specific heat capacity of reaction medium c_p	0.231 kJ kg^{-1}K^{-1}
Order of reaction n	2
Flow rate F	5 m^3/h
Reactor volume V	1 m^3
Integration time step Δt	1/24 s
Feed temperature T_{A0}	300 K

The tank reactor is controlled using an optimization-based controller that ensures that the reactant concentration and temperature reach pre-determined steady-state values, $C_{AS} = 2$ kmol/m^3, $T_S = 400$ K. Equation (16) is used to predict future states with the explicit Euler method, and an optimization problem is set up to minimize the sum of the squared errors for temperature and concentration with respect to the steady-state temperature and concentration of the reactor. The problem is solved using the SciPy [66] `minimize` function with the L-BFGS-B method over a prediction horizon of 10 sampling times. Each initial guess provided to the optimizer is $C_{A0S} = 4$ kmol/m^3, $Q_S = 0$ kJ/h, and the control actions are bounded, with $0.5\ \mathrm{kmol/m^3} \leq C_{A0} \leq 7.5\ \mathrm{kmol/m^3}$, and $-10^{-5}\ \mathrm{kJ/h} \leq Q \leq 10^5\ \mathrm{kJ/h}$.

3.3.2. CSTR Model with Sunlight Factor

In this final investigation of the potential uses of Blender as an image-based testbed for process systems engineering applications, incident sunlight on a reactor that is not insulated is assumed to affect the process primarily by adding heat to the system. It is assumed that the model of the tank reactor is identified at a particular sunlight strength, and hence, when the ambient sunlight is below or above this reference level, heat is removed or added to the system, respectively. In this simulation, the detected sunlight is considered to have an immediate impact on the rate of heat transferred to the system (i.e., potential time lags associated with heat transfer through the material of the tank or mixing within the tank are not modeled). Specifically, Equation (16) is modified by replacing the heat supply variable Q_c with $\hat{Q}_c$, the apparent heat supply, as in Equation (17):

$$\hat{Q}_c = Q_c + \Delta S \epsilon \tag{17}$$

The deviation from the reference sunlight strength is characterized by ΔS and is scaled to the heat supply by a factor ϵ. For this experiment, the reference lighting strength parameter in Blender is chosen to be 2, and the heating factor is set to 5×10^3 kJ/h. While the process model includes the lighting strength directly, the controller utilizes the detected images by correlating the RGB sums to the lighting strength, as detailed in the following section.

3.3.3. Ambient Light Simulation and Detection

In this simulation, images of a reactor and its surroundings are captured using a virtual camera. The intensity of the lighting in the simulation (i.e., the lighting strength) is modified in order to consider changing amounts of sunlight during the day. The model of the reactor is modified so that the heat extracted from the reactor is impacted by the incident light. One of the questions for creating a simulation involving the environment in Blender is how best to set up the lighting source. One method for setting up a lighting source is to use a high dynamic range image (HDRI). An HDRI is used in this simulation to attempt to generate images that resemble outdoor camera sensors. HDRIs are images used for the backgrounds in 3D modeling that wrap around objects in order to provide lighting and environment. The HDRI used in this work, `city.exr`, was obtained from Blender.

Another key aspect of the weather detection simulation is to conceptualize how to detect the sunlight strength from images so that the controller can use the data to update its model. In this example, we consider that the sum of the values of the red, green, and blue channels (RGB sums) are related to the set strength of the lighting (the relationship used in this simulation is calibrated using data from Blender). These RGB sums are calculated by importing the images with Pillow [36] and extracting the color channel values. A set of reference images between the minimum and maximum lighting strengths for the simulation are selected, and then the relation between the natural logarithm of the lighting strength and the RGB sums is plotted, as in Figure 17. RGB sums from the captured images are then used to acquire the lighting strength by interpolating between the known points. Equation (18) details the relationship obtained from the data, where r refers to the RGB sum, S is the lighting strength, and subscripts 1 and 2 refer to the two nearest data points used to interpolate the detected lighting strength:

$$S = S_1 \cdot \left(\frac{S_2}{S_1}\right)^{\frac{r-r_1}{r_2-r_1}} \tag{18}$$

This calibration curve is specific to the chosen background, the resolution of the camera sensor, and the material of the object used to model the CSTR in Blender. Higher image resolutions correlate to higher sums, and an average of the sums could be used to mitigate these effects, subject to the accuracy required. The material of the objects affect the RGB sums by way of their reflectivity, with coarser and finer textures responding differently to the incident light.

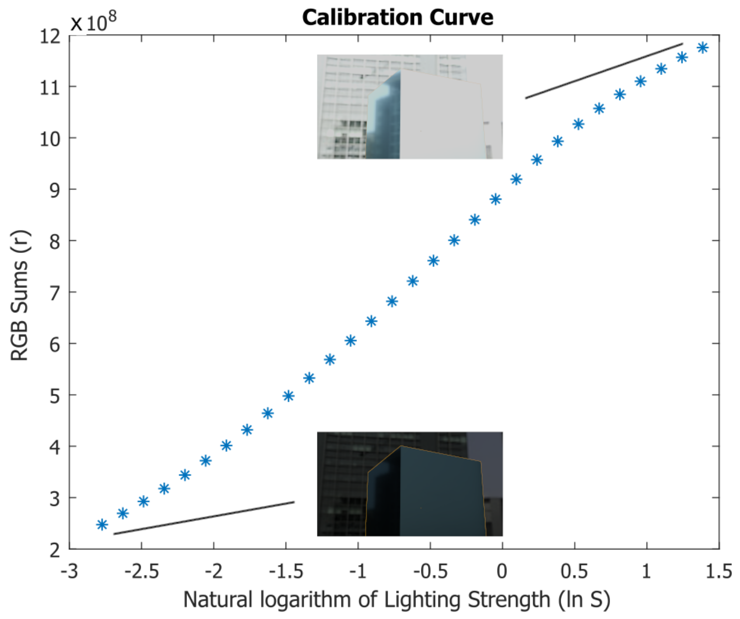


Figure 17. Lighting strength vs RGB sums plot. The yellow outline around the cube in the bottom left reflects that the cube is selected in Blender.

Remark 3. *The lighting strength parameter of HDRIs in Blender allows changes to the lighting model but is more of an artistic rather than a physical parameter. Thus, selecting the lighting strength for an actual application would likely require a comparison of images from an actual plant compared to the virtual model to attempt to find lighting strength values in Blender that match the observed plant behavior. However, even without a data-driven selection of lighting strength, simulations of the type in this work can still be used to test overall controller modification concepts in response to weather variations, knowing that they may need to be fine-tuned once actual process data is available to aid with updating the model. In practice, the lighting strength is replaced by a measurable quantity that is used to calibrate the image sensor. It may also be possible to determine the reference exclusively in terms of images by comparing the sum of the RGB channels of a reference image where the model behaves as designed and assigning a factor that scales the difference in RGB sums between the reference and images captured online to the apparent heat supply Q_c.*

3.3.4. Animation and Control Execution

To investigate the impact of the use of the image-based weather change detection algorithm on the effectiveness of the control policy, simulations were run for a total of 300 s, with each period of constant lighting strength lasting 75 s. In this simulation, lighting strength changes are implemented as step changes to observe the robustness of the controller when weather effects are accounted for. In practice, sunlight strength may vary on a time scale much slower than the controller sampling period. The sunlight was varied across three intensities, i.e., 0.5, 2.0, and 4.0, with time in this simulation, and the specific sunlight variation used in the simulation is detailed in a graph in Figure 18, along with example images captured by the virtual camera.

The differential equations in Equation (16) are used to simulate the state of the CSTR using the explicit Euler method with an integration step size of 1/24 s. The model used to update the reactor includes the lighting strength directly used in Blender. The factor of 1/24 is chosen because the animation runs at 24 frames per second, and the sampling and control is applied every second. The model built into the control optimizer differs from the one used to simulate the process by including the calibrator described in Section 3.3.4.

The sums of the RGB channels of the image are measured by capturing an image from the virtual camera in Blender and then loading it back into the script with the Python Pillow library [36]. The sum calculated is then interpolated to acquire the lighting strength from Equation (18) and then is used in the predictions of the optimizer to acquire an optimized input for the prediction horizon.

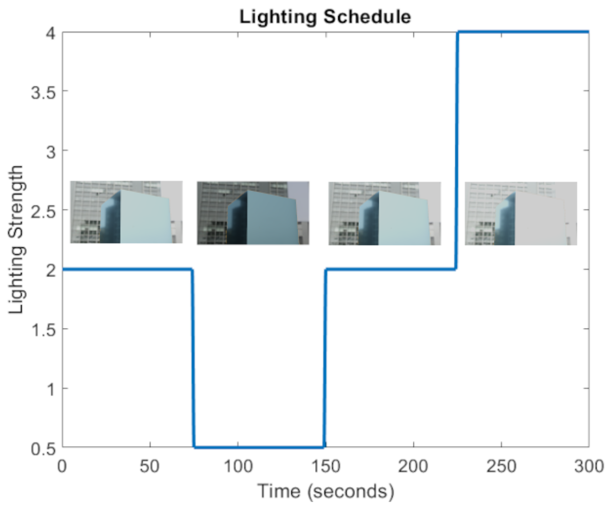


Figure 18. Schedule of lighting strength during simulation, with example images.

3.3.5. Control Effectiveness with and without Weather Detection

In order to compare the effects of light detection on the overall controller performance, the closed-loop responses of the system are compared for the cases where light is accounted for and where it is not accounted for. This is carried out by using Equation (16) for both cases. Figure 19 shows the differing control actions applied in each scenario. The major difference between them is the tracking of reactor temperature. When light detection is active, there is a small disturbance as the optimizer adjusts for a new lighting strength, while when it is inactive, there is a small step change. This also corresponds to the lighting strength in the schedule, with the reactor temperature dipping when the lighting strength is low and rising when it is high. The concentration profiles are largely similar, with differences being extremely small. For the case described, this Blender simulation would demonstrate that an image sensor to detect disturbances in sunlight does not significantly improve the performance of the outdoor reactor on the timescale simulated. This could be useful in determining whether to add such a sensor or not before building one. However, to more comprehensively make such a conclusion, the simulation should be run over a longer time period. Furthermore, longer periods of different sunlight strengths could affect the result.

Table 5 details the lighting strengths set in the simulation and the values detected from the calibration curve using RGB sums of the captured images. Since images with constant lighting strength are identical, errors are displayed for the three lighting strengths in the schedule. This demonstrates that the calibration curve is highly accurate if enough data points are provided. However, this is also subject to disturbances in real world scenarios, where image artifacts and the presence of uncalibrated objects, e.g., operators working in front of a camera sensor, can affect the calibration and interpolation in different ways. However, if a possible disturbance can be modeled with an object in Blender, it can be used to design a calibration curve that meets the requirement of the process. In order to function as a test bed, the calibration of the image sensor requires a more rigorous approach than that demonstrated in this example. Since Blender is built primarily for animation purposes, the lighting strength is an arbitrary unit. Hence, in order to more closely mimic the real world, experimental data that measures deviations from the heat supply and images of the reactor captured by the sensor during that time may be acquired to develop a calibration metric.

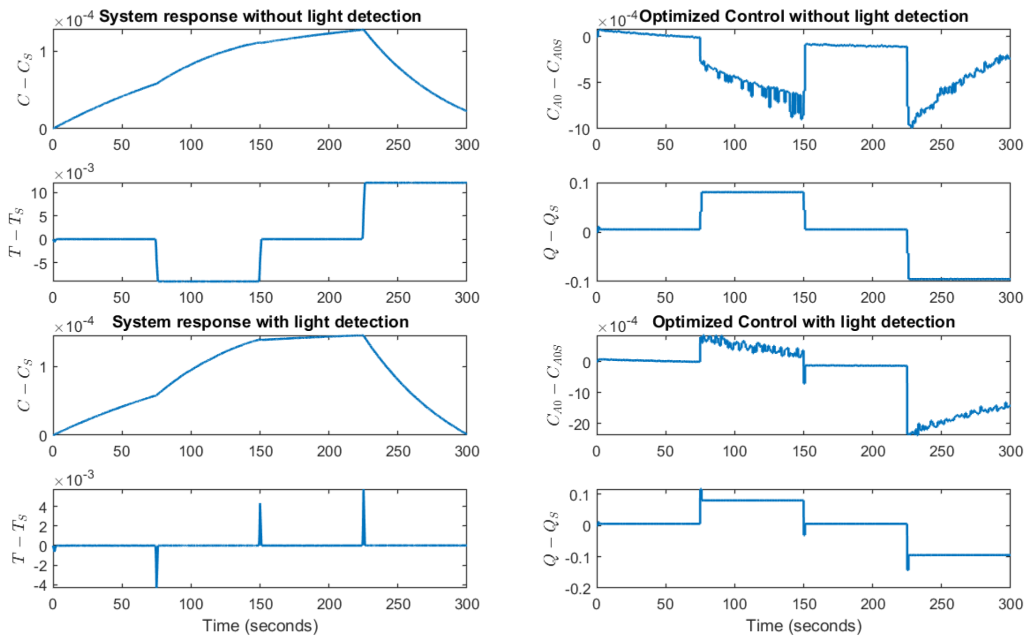


Figure 19. Comparison of system response and optimal control actions when light detection is active and inactive. All variables are plotted as deviations from steady-state values. Units: concentration (C, C_{A0})—kmol/m^3, temperature (T)—K, heat (Q)—kJ/h.

Table 5. Errors in detecting scheduled lighting strengths (see Remark 3 for a discussion of lighting strength in Blender).

Simulation Lighting Strength	Detected Lighting Strength
0.5	0.5001
2.0	2.0025
4.0	4.0000

Remark 4. *The general concept of utilizing Blender to test different concepts for weather detection for updating optimization-based control laws can be applied to other weather phenomena as well (e.g., rain). Though this work has discussed a number of potential challenges and opportunities with respect to using Blender for image-based control tests, many other considerations remain to be investigated, including how to make more photo-realistic models to ensure that the types of tests performed are representative of what might be seen in the field. In general, the concept that auxiliary data that is not typically used in updating controller models might be able to aid in knowing when to update them (i.e., revealing that a physics-based model is incomplete (e.g., it may not account for the impacts of weather variations on the process dynamics) can also be potentially applied with other data (e.g., there is potential that if anomalies are detected in the state data [67–69] that causes for these might be embedded in data such as auxiliary process numerical/textual information [70–72] (e.g., weather or maintenance logs) to seek to identify conditions that may have caused the modeling discrepancies and then to develop methods for detecting when such conditions are about to occur (e.g., utilizing pattern recognition [73,74] to note trends in the process data similar to those leading up to the prior occurrence of the anomaly).*

4. Conclusions

In this work, we demonstrate the potential of using Blender as a simulation test bed for image-based control with a few example processes. First, physical phenomena are replicated in Blender utilizing limited data and relying on the interpolation of object properties as a substitute to numerical integration. The motion of a bubble moving through a liquid is simulated with a select number of forces, and the position over time is acquired and animated in Blender using keyframe information. The position values acquired from Blender between keyframes where they are explicitly defined are compared with the results of numerical integration carried out at a finer timescale. Second, Blender is successfully used to replicate Brownian motion dynamics of the nanorod in two-dimensional space, and the optimal stochastic policy is implemented using a virtual image-based controller. Keyframe features are leveraged to demonstrate the scaling of time between the simulation and the control algorithm, where Brownian motion dynamics were updated frame by frame, but keyframes were inserted and images were captured every second to enact control. A virtual camera and light source are used to simulate the image-based control policy. The color information is summed and is used to detect the rod and extract information about its state features. Image capture by the software and additional image processing with the Python Pillow library are demonstrated, and the error between the state of the system from Blender and states detected in the image-based control algorithm is ascertained. Finally, a simulation of sunlight as a disturbance in processes that can be affected by ambient conditions is carried out in Blender. Utilizing the HDRI lighting settings, an outdoor environment is simulated, and the strength of the lighting is varied to simulate changes in daylight. These settings are then calibrated with the sum of the red, blue, and green channels of the images captured by the virtual camera in Blender to approximate lighting strength based on a given image. The process is then simulated with a controller that optimizes for steady-state behavior while also accounting for varying sunlight. A comparison of the responses to the process based on whether the controller accounts for the changes in sunlight is illustrated. With the framework currently in place, it opens the possibility of virtually testing these and other image processing techniques in order to rapidly test novel image-based control algorithms in a variety of processes. Future studies could include further investigations into the potential of Blender for use in developing image-based control and safety systems.

Author Contributions: A.F.L.: conceptualization, methodology, software, validation G.G.: software, methodology M.R.: conceptualization, methodology H.E.D.: supervision, funding acquisition, validation. All authors have read and agreed to the published version of the manuscript.

Funding: Financial support from the Air Force Office of Scientific Research (award number FA9550-19-1-0059), National Science Foundation CNS-1932026 and CBET-1839675, and Wayne State University is gratefully acknowledged.

Data Availability Statement: The codes associated with this article will be posted at: https://durand.eng.wayne.edu/PublicationResources/index.html accessed on 2 September 2023.

Acknowledgments: We would like to thank Michael Williamson, Keshav Kasturi Rangan, and Henrique Oyama for the discussions, and Jacob Noll for finding reference [39]. Figures in this publication have been created with Blender versions 3.6 and 4.0, MATLAB R2023a, GNU Image Manipulation Program 2.10.36, and draw.io v22.1.8.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Prats-Montalbán, J.M.; de Juan, A.; Ferrer, A. Multivariate image analysis: A review with applications. *Chemom. Intell. Lab. Syst.* **2011**, *107*, 1–23. [CrossRef]
2. De, S.; Mohamed, S.; Bimpisidis, K.; Goswami, D.; Basten, T.; Corporaal, H. Approximation trade offs in an image-based control system. In Proceedings of the 2020 Design, Automation & Test in Europe Conference & Exhibition (DATE), Grenoble, France, 9–13 March 2020; IEEE: Piscataway, NJ, USA, 2020; pp. 1680–1685.
3. Bengler, K.; Dietmayer, K.; Farber, B.; Maurer, M.; Stiller, C.; Winner, H. Three decades of driver assistance systems: Review and future perspectives. *IEEE Intell. Transp. Syst. Mag.* **2014**, *6*, 6–22. [CrossRef]
4. Su, Y.; Zheng, C. A simple PID control for asymptotic visual regulation of robot manipulators. *Int. J. Robust Nonlinear Control* **2011**, *21*, 1525–1540. [CrossRef]
5. Bluma, A.; Höpfner, T.; Lindner, P.; Rehbock, C.; Beutel, S.; Riechers, D.; Hitzmann, B.; Scheper, T. In-situ imaging sensors for bioprocess monitoring: State of the art. *Anal. Bioanal. Chem.* **2010**, *398*, 2429–2438. [CrossRef]
6. Höpfner, T.; Bluma, A.; Rudolph, G.; Lindner, P.; Scheper, T. A review of non-invasive optical-based image analysis systems for continuous bioprocess monitoring. *Bioprocess Biosyst. Eng.* **2010**, *33*, 247–256. [CrossRef]
7. Chandrasekaran, S.N.; Ceulemans, H.; Boyd, J.D.; Carpenter, A.E. Image-based profiling for drug discovery: Due for a machine-learning upgrade? *Nat. Rev. Drug Discov.* **2021**, *20*, 145–159. [CrossRef]

8. Mebarki, R.; Lippiello, V. Image-based control for aerial manipulation. *Asian J. Control* **2014**, *16*, 646–656. [CrossRef]
9. Collewet, C.; Chaumette, F. A contour approach for image-based control on objects with complex shape. In Proceedings of the 2000 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2000) (Cat. No. 00CH37113), Takamatsu, Japan, 31 October–5 November 2000; IEEE: Piscataway, NJ, USA, 2000; Volume 1, pp. 751–756.
10. Zheng, D.; Wang, H.; Wang, J.; Chen, S.; Chen, W.; Liang, X. Image-based visual servoing of a quadrotor using virtual camera approach. *IEEE/ASME Trans. Mechatron.* **2016**, *22*, 972–982. [CrossRef]
11. Becker, B.C.; MacLachlan, R.A.; Lobes, L.A.; Hager, G.D.; Riviere, C.N. Vision-based control of a handheld surgical micromanipulator with virtual fixtures. *IEEE Trans. Robot.* **2013**, *29*, 674–683. [CrossRef] [PubMed]
12. Pazzi, R.W.N.; Boukerche, A.; Huang, T. Implementation, measurement, and analysis of an image-based virtual environment streaming protocol for wireless mobile devices. *IEEE Trans. Instrum. Meas.* **2008**, *57*, 1894–1907. [CrossRef]
13. Lee, D.; Lim, H.; Kim, H.J. Obstacle avoidance using image-based visual servoing integrated with nonlinear model predictive control. In Proceedings of the 2011 50th IEEE Conference on Decision and Control and European Control Conference, Orlando, FL, USA, 12–15 December 2011; IEEE: Piscataway, NJ, USA, 2011; pp. 5689–5694.
14. Hajiloo, A.; Keshmiri, M.; Xie, W.F.; Wang, T.T. Robust online model predictive control for a constrained image-based visual servoing. *IEEE Trans. Ind. Electron.* **2015**, *63*, 2242–2250.
15. Lopez-Franco, C.; Gomez-Avila, J.; Alanis, A.Y.; Arana-Daniel, N.; Villaseñor, C. Visual servoing for an autonomous hexarotor using a neural network based PID controller. *Sensors* **2017**, *17*, 1865. [CrossRef]
16. Yu, H.; MacGregor, J.F. Monitoring flames in an industrial boiler using multivariate image analysis. *AIChE J.* **2004**, *50*, 1474–1483. [CrossRef]
17. Lin, B.; Recke, B.; Knudsen, J.K.; Jørgensen, S.B. Bubble size estimation for flotation processes. *Miner. Eng.* **2008**, *21*, 539–548. [CrossRef]
18. Cao, Y.; Yu, H.; Abbott, N.L.; Zavala, V.M. Machine learning algorithms for liquid crystal-based sensors. *ACS Sens.* **2018**, *3*, 2237–2245. [CrossRef] [PubMed]
19. Pulsipher, J.L.; Coutinho, L.D.J.; Soderstrom, T.A.; Zavala, V.M. SAFE-OCC: A novelty detection framework for Convolutional Neural Network sensors and its application in process control. *J. Process Control* **2022**, *117*, 78–97. [CrossRef]
20. Jiang, S.; Qin, S.; Pulsipher, J.L.; Zavala, V.M. Convolutional Neural Networks: Basic Concepts and Applications in Manufacturing. *arXiv* **2022**, arXiv:2210.07848.
21. Wang, X.Z.; Roberts, K.J.; Ma, C. Crystal growth measurement using 2D and 3D imaging and the perspectives for shape control. *Chem. Eng. Sci.* **2008**, *63*, 1173–1184. [CrossRef]
22. Larsen, P.A.; Rawlings, J.B.; Ferrier, N.J. An algorithm for analyzing noisy, in situ images of high-aspect-ratio crystals to monitor particle size distribution. *Chem. Eng. Sci.* **2006**, *61*, 5236–5248. [CrossRef]
23. Maaß, S.; Rojahn, J.; Hänsch, R.; Kraume, M. Automated drop detection using image analysis for online particle size monitoring in multiphase systems. *Comput. Chem. Eng.* **2012**, *45*, 27–37. [CrossRef]
24. Chen, J.; Chang, Y.H.; Cheng, Y.C.; Hsu, C.K. Design of image-based control loops for industrial combustion processes. *Appl. Energy* **2012**, *94*, 13–21. [CrossRef]
25. Lu, Q.; Zavala, V.M. Image-based model predictive control via dynamic mode decomposition. *J. Process Control* **2021**, *104*, 146–157. [CrossRef]
26. Pearson, T.; Brabec, D.; Haley, S. Color image based sorter for separating red and white wheat. *Sens. Instrum. Food Qual. Saf.* **2008**, *2*, 280–288. [CrossRef]
27. Dere, S.; Sahasrabudhe, S.; Iyer, S. Creating open source repository of 3D models of laboratory equipments using Blender. In Proceedings of the 2010 International Conference on Technology for Education, Mumbai, India, 1–3 July 2010; pp. 149–156.
28. Rajendiran, N.; Durrant, J.D. Pyrite: A Blender plugin for visualizing molecular dynamics simulations using industry-standard rendering techniques. *J. Comput. Chem.* **2018**, *39*, 748–755. [CrossRef]
29. Gschwandtner, M.; Kwitt, R.; Uhl, A.; Pree, W. BlenSor: Blender sensor simulation toolbox. In *Proceedings of the Advances in Visual Computing: 7th International Symposium, ISVC 2011, Las Vegas, NV, USA, 26–28 September 2011*; Proceedings, Part II 7; Springer: Berlin/Heidelberg, Germany, 2011; pp. 199–208.
30. Flaischen, S.; Wehinger, G.D. Synthetic packed-bed generation for CFD simulations: Blender vs. STAR-CCM+. *ChemEngineering* **2019**, *3*, 52. [CrossRef]
31. Oyama, H.; Leonard, A.F.; Rahman, M.; Gjonaj, G.; Williamson, M.; Durand, H. On-line process physics tests via Lyapunov-based economic model predictive control and simulation-based testing of image-based process control. In Proceedings of the 2022 American Control Conference (ACC), Atlanta, GA, USA, 8–10 June 2022; IEEE: Piscataway, NJ, USA, 2022; pp. 2479–2484.
32. Parzen, E. *Stochastic Processes*; SIAM: Philadelphia, PA, USA, 1999.
33. Øksendal, B. Stochastic differential equations . In *Stochastic Differential Equations*; Springer: Berlin/Heidelberg, Germany, 2003; pp. 65–84.
34. Wrobel, A. On Markovian decision models with a finite skeleton. *Z. Für Oper. Res.* **1984**, *28*, 17–27. [CrossRef]
35. Gangwani, T.; Li, D.; Ye, Z. Lecture 16: Value Iteration, Policy Iteration and Policy Gradient. Available online: https://yuanz.web.illinois.edu/teaching/IE498fa19/lec_16.pdf (accessed on 4 September 2023).
36. Clark, A. Pillow (PIL Fork) Documentation. ReadTheDocs. 2015. Available online: https://buildmedia.readthedocs.org/media/pdf/pillow/latest/pillow.pdf (accessed on 4 September 2023).

37. Oyama, H.; Messina, D.; O'Neill, R.; Cherney, S.; Rahman, M.; Rangan, K.K.; Gjonaj, G.; Durand, H. Test Methods for Image-Based Information in Next-Generation Manufacturing. *IFAC-PapersOnLine* **2022**, *55*, 73–78. [CrossRef]
38. Oyama, H.; Messina, D.; Rangan, K.K.; Leonard, A.F.; Nieman, K.; Durand, H.; Tyrrell, K.; Hinzman, K.; Williamson, M. Development of directed randomization for discussing a minimal security architecture. *Digit. Chem. Eng.* **2023**, *6*, 100065. [CrossRef]
39. Yang, Y. Stochastic Modeling and Optimal Control for Colloidal Organization, Navigation, and Machines. Ph.D. Thesis, The Johns Hopkins University, Baltimore, MD, USA, 2017.
40. Bartolacci, G.; Pelletier, P.; Tessier, J.; Duchesne, C.; Bossé, P.A.; Fournier, J. Application of numerical image analysis to process diagnosis and physical parameter measurement in mineral processes—Part I: Flotation control based on froth textural characteristics. *Miner. Eng.* **2006**, *19*, 734–747. [CrossRef]
41. Kaartinen, J. ; Hätönen, J.; Hyötyniemi, H.; Miettunen, J. Machine-vision-based control of zinc flotation—A case study. *Control Eng. Pract.* **2006**, *14*, 1455–1466. [CrossRef]
42. Liu, J.J.; MacGregor, J.F.; Duchesne, C.; Bartolacci, G. Flotation froth monitoring using multiresolutional multivariate image analysis. *Miner. Eng.* **2005**, *18*, 65–76. [CrossRef]
43. Bharati, M.H.; MacGregor, J.F. Multivariate image analysis for real-time process monitoring and control. *Ind. Eng. Chem. Res.* **1998**, *37*, 4715–4724. [CrossRef]
44. Bird, R.B.; Stewart, W.E.; Lightfoot, E.N. *Transport Phenomena*; John Wiley & Sons: Hoboken, NJ, USA, 1961.
45. Cleary, P.W.; Pyo, S.H.; Prakash, M.; Koo, B.K. Bubbling and frothing liquids. In *ACM SIGGRAPH 2007 Papers*; ACM: New York, NY, USA, 2007; p. 97-es.
46. Dewitt, C.C. Froth flotation concentration. *Ind. Eng. Chem.* **1940**, *32*, 652–658. [CrossRef]
47. Cartesian Caramel. How to Make Oil Spill Materials in Blender! Video, Uploaded to Youtube. 20 February 2022. Available online: https://youtu.be/xcx_LfXuuX4 (accessed on 31 October 2023).
48. Do, H. Development of a Turbulent Flotation Model from First Principles. Ph.D. Thesis, Virginia Tech, Blacksburg, VA, USA, 2010.
49. Lin, C.; Liu, Y.; Rinker, S.; Yan, H. DNA tile based self-assembly: Building complex nanoarchitectures. *ChemPhysChem* **2006**, *7*, 1641–1647. [CrossRef]
50. Douglas, S.M.; Dietz, H.; Liedl, T.; Högberg, B.; Graf, F.; Shih, W.M. Self-assembly of DNA into nanoscale three-dimensional shapes. *Nature* **2009**, *459*, 414–418. [CrossRef]
51. Wang, X.S.; Winnik, M.A.; Manners, I. Synthesis and aqueous self-assembly of a polyferrocenylsilane-block-poly (aminoalkyl methacrylate) diblock copolymer. *Macromol. Rapid Commun.* **2002**, *23*, 210–213. [CrossRef]
52. Rider, D.A.; Manners, I. Synthesis, Self-Assembly, and Applications of Polyferrocenylsilane Block Copolymers. *Polym. Rev.* **2007**, *47*, 165–195. [CrossRef]
53. Liu, R.; Huang, H.; Sun, Z.; Alexander-Katz, A.; Ross, C.A. Metallic nanomeshes fabricated by multimechanism directed self-assembly. *ACS Nano* **2021**, *15*, 16266–16276. [CrossRef]
54. Lane, A.P.; Yang, X.; Maher, M.J.; Blachut, G.; Asano, Y.; Someya, Y.; Mallavarapu, A.; Sirard, S.M.; Ellison, C.J.; Willson, C.G. Directed self-assembly and pattern transfer of five nanometer block copolymer lamellae. *ACS Nano* **2017**, *11*, 7656–7665. [CrossRef]
55. Duan, H.; Berggren, K.K. Directed self-assembly at the 10 nm scale by using capillary force-induced nanocohesion. *Nano Lett.* **2010**, *10*, 3710–3716. [CrossRef]
56. Yu, J.; Wang, Q.; Zhang, X. Effects of external force fields on peptide self-assembly and biomimetic silica synthesis. *Appl. Surf. Sci.* **2014**, *311*, 799–807. [CrossRef]
57. Motornov, M.; Malynych, S.Z.; Pippalla, D.S.; Zdyrko, B.; Royter, H.; Roiter, Y.; Kahabka, M.; Tokarev, A.; Tokarev, I.; Zhulina, E.; et al. Field-directed self-assembly with locking nanoparticles. *Nano Lett.* **2012**, *12*, 3814–3820. [CrossRef] [PubMed]
58. Tang, X.; Rupp, B.; Yang, Y.; Edwards, T.D.; Grover, M.A.; Bevan, M.A. Optimal feedback controlled assembly of perfect crystals. *ACS Nano* **2016**, *10*, 6791–6798. [CrossRef] [PubMed]
59. Issa, H.; Natale, G.; Ausias, G.; Férec, J. Modeling and numerical simulations of Brownian rodlike particles with anisotropic translational diffusion. *Phys. Rev. Fluids* **2023**, *8*, 033302. [CrossRef]
60. Durrett, R. *Probability: Theory and Examples*; Cambridge Series in Statistical and Probabilistic Mathematics; Cambridge University Press: Cambridge, UK, 2019.
61. Alexandru, A.B.; Morari, M.; Pappas, G.J. Cloud-based MPC with encrypted data. In Proceedings of the 2018 IEEE Conference on Decision and Control (CDC), Miami, FL, USA, 17–19 December 2018; pp. 5014–5019.
62. Zavala, V.M.; Constantinescu, E.M.; Krause, T.; Anitescu, M. On-line economic optimization of energy systems using weather forecast information. *J. Process Control* **2009**, *19*, 1725–1736. [CrossRef]
63. Rowe, S.C.; Hischier, I.; Palumbo, A.W.; Chubukov, B.A.; Wallace, M.A.; Viger, R.; Lewandowski, A.; Clough, D.E.; Weimer, A.W. Nowcasting, predictive control, and feedback control for temperature regulation in a novel hybrid solar-electric reactor for continuous solar-thermal chemical processing. *Sol. Energy* **2018**, *174*, 474–488. [CrossRef]
64. Das, B.; Mhaskar, P. Lyapunov-based offset-free model predictive control of nonlinear process systems. *Can. J. Chem. Eng.* **2015**, *93*, 471–478. [CrossRef]
65. Alanqar, A.; Durand, H.; Christofides, P.D. Fault-tolerant economic model predictive control using error-triggered online model identification. *Ind. Eng. Chem. Res.* **2017**, *56*, 5652–5667. [CrossRef]

66. Virtanen, P.; Gommers, R.; Oliphant, T.E.; Haberland, M.; Reddy, T.; Cournapeau, D.; Burovski, E.; Peterson, P.; Weckesser, W.; Bright, J.; et al. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nat. Methods* **2020**, *17*, 261–272. [CrossRef]
67. Lane, T.; Brodley, C.E. An application of machine learning to anomaly detection. In Proceedings of the 20th National Information Systems Security Conference, Baltimore, MD, USA, 7–10 October 1997.
68. Shon, T.; Moon, J. A hybrid machine learning approach to network anomaly detection. *Inf. Sci.* **2007**, *177*, 3799–3821. [CrossRef]
69. Vernekar, S.; Nari, S.; Vijaysenan, D.; Ranjan, R. A novel approach for classification of normal/abnormal phonocardiogram recordings using temporal signal analysis and machine learning. In Proceedings of the 2016 Computing in Cardiology Conference, Vancouver, BC, Canada, 11–14 September 2016; Volume 63, pp. 1141–1144.
70. Turney, P.D.; Pantel, P. From frequency to meaning: Vector space models of semantics. *J. Artif. Intell. Res.* **2010**, *37*, 141–188. [CrossRef]
71. Zhang, Y.; Jiang, R.; Petzold, L. Survival Topic Models for Predicting Outcomes for Trauma Patients. In Proceedings of the IEEE 33rd International Conference on Data Engineering, San Diego, CA, USA, 19–22 April 2017; pp. 1497–1504.
72. Nguyen, N.T.H.; Miwa, M.; Tsuruoka, Y.; Tojo, S. Identifying synonymy between relational phrases using word embeddings. *J. Biomed. Inform.* **2015**, *56*, 94–102. [CrossRef] [PubMed]
73. Lopes, A.T.; de Aguiar, E.; De Souza, A.F.; Oliveira-Santos, T. Facial expression recognition with convolutional neural networks: Coping with few data and the training sample order. *Pattern Recognit.* **2017**, *61*, 610–628. [CrossRef]
74. Uhlmann, E.; Pontes, R.P.; Laghmouchi, A.; Bergmann, A. Intelligent pattern recognition of a SLM machine process and sensor data. *Procedia CIRP* **2017**, *62*, 464–469. [CrossRef]