# Ma Ph 499: Final Report

## Generating Artificial Event Images from the PICO-60 Dark Matter Experiment

Adam Smith

April 8, 2020

# I Abstract

The PICO-60 dark matter detector is a bubble chamber experiment filled with liquid $C_3F_8$ operated at the SNOLAB underground laboratory searching for direct evidence of WIMP dark matter particles. If a sufficient amount of training examples are available, artificial neural networks offer valuable tools for data analysis on such detectors. However, due to the low-background design of PICO-60 and similar detectors, data on which to train neural network models is often limited. In this report, we devise a method for generating artificial optical data mimicking images obtained from the PICO-60 detector's four cameras. Existing event images are analyzed to remove background and isolate bubbles, and an unsupervised generative modeling framework called a generative adversarial network (GAN) is applied to produce artificial bubble images. A ray tracing algorithm is then applied to compute pixel coordinates of bubbles in each camera given a random bubble position within the detector's active volume, and a set of four images is composed, representing a virtual PICO-60 event. The distributions of pixel coordinates for each of the four cameras are compared between artificial and PICO-60 images, and Pearson correlation coefficients of 0.83, 0.81, 0.81, and 0.79 are obtained for cameras 0, 1, 2, and 3 respectively. Bubble radius is also compared for artificial and real images, and a similar relationship between bubble radius and distance from the camera is observed between datasets.

# II Introduction and Theory

The PICO-60 experiment, operated in 2016 and 2017 at the SNOLAB underground laboratory, is a bubble chamber experiment aimed at detection of WIMP dark matter particles. It utilizes superheated liquid freon with the aim of observing nuclear recoils produced by WIMP-nucleon scattering, and is the largest bubble chamber to search for dark matter to date [1]. PICO-60 exhibits excellent intrinsic rejection of gamma and beta events, and excellent alpha rejection through acoustic information. Multiple scattering neutron events are primarily rejected through topological properties.

## II.I PICO-60 Design

The PICO-60 bubble chamber, shown in Fig. 1 and described in Ref. [1], consists of a 30-cm-diameter by 1-m-long fused silica cylindrical jar with a dome-shaped end cap. The jar contains 52 kg of superheated liquid $C_3F_8$ as a target volume, and is sealed to flexible stainless-steel bellows and immersed in hydraulic fluid, all contained within a stainless-steel pressure vessel. The pressure vessel is submerged in a water

tank to provide shielding from external radiation sources and temperature control. The bellows balance the pressure between the hydraulic fluid and the target liquid of the bubble chamber, and a water buffer layer sits on top of the $C_3F_8$ to isolate the target liquid from contact with stainless-steel surfaces.

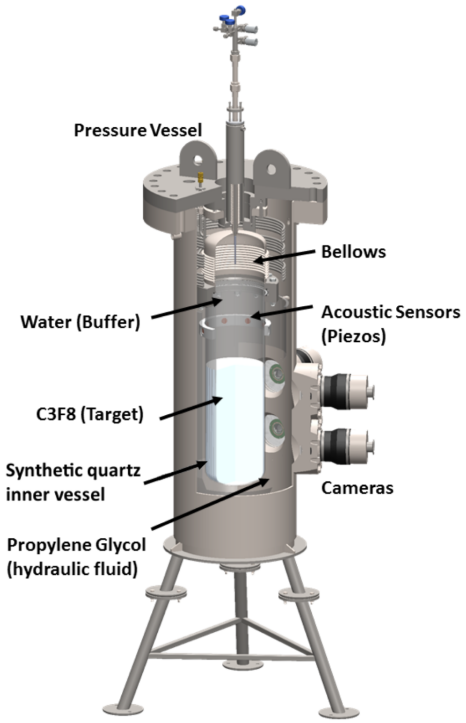


Figure 1: Schematic of the PICO-60 detector. The detector stands roughly 2 m tall and is immersed in a large tank containing ultrapure water. Source: Ref. [2].

The active volume of the chamber is imaged by two columns of two 1088×1700 high-speed CMOS cameras. The appearance of bubbles is continuously monitored by calculating the spatial temporal image entropy

$$S_I = \sum_i P_i \log_2 P_i \tag{1}$$

of the absolute difference between consecutive frames. The values $P_i$ are the fractions of pixels populating intensity bin $i$ in the difference image histogram generated by consecutive frames [2]. If $S_I$ exceeds an empirically determined threshold in any of the cameras, a trigger signal is sent by the camera system to indicate an event has occurred. This initiates the recompression of the bubble chamber and data associated with the event is saved. Included in this data is 40 frames from each camera surrounding the trigger. An example of the trigger frame for two of the cameras in a 17-bubble event is shown in Fig. 2.

## II.II Machine Learning for Dark Matter Searches

In recent years, machine learning algorithms, most notably neural networks, have seen much success in a wide variety of disciplines. In particular, neural networks have been applied to object recognition and image reconstruction, often outperforming traditional automated methods [3, 4]. However, a major drawback of machine learning techniques is the need for large amounts of training data to facilitate effective training of the network. A network which lacks training examples has difficulty learning a valid model and is prone to overfitting [5].

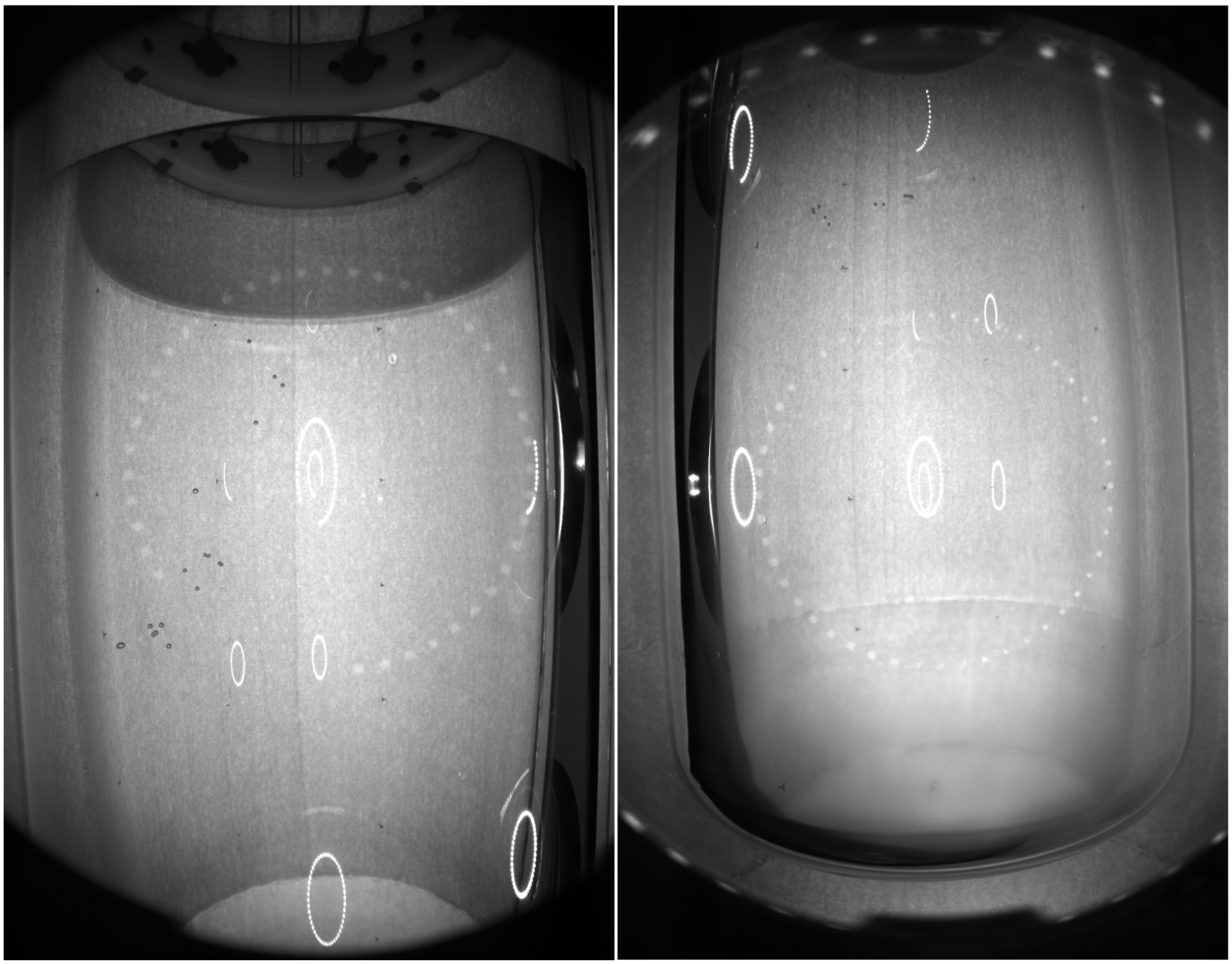

Figure 2: Images from camera 0 (left) and camera 3 (right) of the trigger frame of a 17 bubble event from a calibration run.

Due to the low-background design of the PICO-60 detector and other dark matter search experiments, examples on which to train neural networks for data analysis are limited. Moreover, multiple data channels are obtained from detected events. Pressure, optical, and acoustic data must be analyzed collectively to provide a complete analysis. These factors make obtaining effective neural network models for tasks such as 3D position reconstruction of bubbles difficult.

In this report, a method for generating artificial images of events in PICO-60 and similar detectors is developed. We employ a generative modeling neural network architecture to produce artificial bubble images and compose them into sets of four images representing optical data from a virtual event. This algorithm provides a dataset on which to train models and employ transfer learning [6] techniques for analysis of real data. Bubble radius and pixel position distributions correspond strongly with those of the PICO-60 data, and training examples are no longer limited in availability.

## II.III Feedforward Neural Networks

An artificial neural network is a function $\mathcal{N} : \mathbb{R}^N \to \mathbb{R}^M$ containing several adjustable parameters which maps input data $\mathbf{x} \in \mathbb{R}^N$ to an output $\mathbf{y} \in \mathbb{R}^M$. The network's output is then compared to known correct

outputs, and the parameters of the neural network are adjusted to fit the training dataset. This allows for automated generation of complex models describing a wide variety of processes.

A feedforward neural network is built by composition of several layers, each an affine transformation with a corresponding non-linear *activation function*. A weight matrix $W \in \mathbb{R}^{p \times q}$ and bias vector $\mathbf{b} \in \mathbb{R}^p$ are associated to each layer. The layer $l$, called a *fully-connected* layer, is defined as the function

$$l : \mathbb{R}^q \to \mathbb{R}^p, \quad \mathbf{x} \mapsto \phi\left(W\mathbf{x} + \mathbf{b}\right) \tag{2}$$

where $\phi : \mathbb{R} \to \mathbb{R}$ is the activation function and is applied componentwise. One or more layers are then composed to define the neural network $\mathcal{N}$. It is important that the activation function contain a non-linearity. Otherwise, the network is simply a composition of affine transformations, so it is itself affine. A common activation function is the rectified linear unit (ReLU) defined as $f(x) = \max(0, x)$ or its variant, leaky ReLU, define as

$$f(x) = \begin{cases} x & \text{if } x > 0 \\ \gamma x & \text{otherwise} \end{cases} \tag{3}$$

where $\gamma \ll 1$ is a small, positive constant.

In order to generate a model, we must have a procedure for updating the weight matrices and bias vectors within the neural network to fit a set of training data. To do this, we first define a loss function

$$L : \mathbb{R}^M \times \mathbb{R}^M \to \mathbb{R} \tag{4}$$

where $L(\mathbf{y}, \mathbf{z})$ reaches a global minimum when $\mathbf{y} = \mathbf{z}$. A typical loss function is the mean squared error loss function

$$L_{MSE}(\mathbf{y}, \mathbf{z}) = ||\mathbf{y} - \mathbf{z}||^2. \tag{5}$$

Representing the elements of the all weight matrices and bias vectors as a vector of parameters $\theta$, the objective is to minimize

$$Q(\theta; n) = \frac{1}{n} \sum_{i=1}^{n} Q_i(\theta) \tag{6}$$

with respect to the parameters $\theta$ where

$$Q_i(\theta) = L(\mathcal{N}(\mathbf{x}_i; \theta), \mathbf{z}_i) \tag{7}$$

is the loss of the $i^{\text{th}}$ training example, and $\mathbf{x}_i$ is an element of the set of $n$ training examples in the training dataset with corresponding desired output $\mathbf{z}_i$. The simplest method for doing so is through the use of stochastic gradient descent (SGD). SGD approximates the gradient of $Q$ by the mean of the gradients of the losses $Q_i$ for a randomly chosen small subset of $m$ training examples from the training dataset. Since $\theta$ is usually from a high-dimensional space and the total number of training examples $n$ is often large, this step is necessary to reduce computation times for practical usage. The parameters are then updated via the rule

$$\theta \to \theta - \eta \nabla Q(\theta; m). \tag{8}$$

Here $\eta$ is a hyperparameter known as the *learning rate* which controls the step size of each iteration. Setting the learning rate effectively has proven to be a difficult task. Setting it too low results in slow convergence, and setting it too high can cause divergence or inaccurate estimations of the minimum [7]. An improvement on SGD known as *Adam* [8] (short for Adaptive Moment Estimation) adapts the learning rate through

successive iterations using moving averages of the gradient estimation and its square. With $t$ indexing the current training iteration, the parameters are updated via the rule

$$\mathbf{k}_0 = \mathbf{v}_0 = 0$$

$$\mathbf{k}_{t+1} = \beta_1 \mathbf{k}_t + (1-\beta_1)\left(\nabla Q(\theta; m)\right)_t$$
$$\mathbf{v}_{t+1} = \beta_2 \mathbf{v}_t + (1-\beta_2)\left(\nabla Q(\theta; m)\right)_t^2$$
$$\hat{\mathbf{k}} = \frac{\mathbf{k}_{t+1}}{1-\beta_1^{t+1}}$$
$$\hat{\mathbf{v}} = \frac{\mathbf{v}_{t+1}}{1-\beta_2^{t+1}}$$
$$\theta_{t+1} = \theta_t - \eta \frac{\hat{\mathbf{k}}}{\sqrt{\hat{\mathbf{v}}} + \epsilon} \tag{9}$$

where the squares and square roots are applied componentwise, $\epsilon$ is a small scalar to prevent division by zero, and $\beta_1$ and $\beta_2$ are *forgetting factors* for the gradient and its square, respectively. By making use of running averages, Adam is insensitive to abrupt changes in the magnitude of the gradient, making it particularly useful for machine learning applications.

## II.IV Convolutional Layers

Often, it is desirable to obtain a neural network with the ability to train directly on images without first collapsing their matrix representations into vectors. This allows us to better capture the spatial dependencies of values within an image. To do this, we replace the affine transformation in each layer with a linear convolution operation which applies a filter (known as a *kernel*) to the input matrix $A$ to output a matrix $B = (b_{ij})$, typically of smaller size. In convolution, the kernel $K \in \mathbb{R}^{s \times s}$ is defined and its elements are the parameters to be adjusted to train the network. Here $s$ is typically small (usually 3 or 5) but can be as large as the smallest dimension of $A$. The scalar product of the kernel with the upper left $s \times s$ submatrix of $A$ is taken and used to define the element $b_{11}$ of the output matrix. The kernel is then shifted right by a fixed amount known as the *stride* (typically stride is one or two) along the input matrix, and the scalar product of the kernel with the new corresponding $s \times s$ submatrix defines the element $b_{12}$ of the output matrix. This is continued across each row of the input matrix. Finally, the non-linear activation function is applied componentwise to the output matrix.

Several adjustments can be made to tune the convolution operation to the specific application, namely

- A border of zeros (known as *padding*) can be added to the boundary of the input matrix. This is typically done to preserve matrix size after a convolution operation.
- The kernel can be given a *dilation*. This "spreads out" the elements of the kernel, increasing its size and sparsity, and has shown to be particularly useful in semantic segmentation systems [9].
- A matrix of trainable bias parameters can be added to the output matrix in between the convolution and activation function operations.

Typically, multiple convolution operations using different kernels are applied in each convolutional layer. This results in several output matrices (channels) for each input matrix. In further convolutional layers, the convolution operations are applied separately to each input channel.

A convolutional neural network (CNN) is defined by composing one or more convolutional layers. To achieve a vector or scalar output instead of a matrix from a CNN, the output from the last convolutional layer is flattened into a vector and a final fully-connected layer is applied.

### II.V Generative Adversarial Networks

Neural networks have seen much success in problems involving discriminative models, usually those that map a high-dimensional, rich input to a categorical label [10, 11]. However, up until recently, neural networks have had far less success in the generation of generative models. In 2014, a type of neural network architecture called a generative adversarial network (GAN) [12] was proposed as a major breakthrough in generative model estimation. A GAN uses two neural networks, a generator and discriminator, usually both CNNs. The generator produces synthetic samples of the same format to a training dataset, and the discriminator works to discriminate between real samples from the training dataset and those generated by the generator. These networks contest against each other to improve their methods until counterfeit samples are indistinguishable from real samples.

The generator $G(\mathbf{z};\theta_g)$ is feedforward neural network with parameters $\theta_g$ which maps an input $\mathbf{z}$ sampled randomly from a prior $p_{\mathbf{z}}(\mathbf{z})$ on the latent space to the space of the training data. Often, $\mathbf{z}$ is a vector with elements drawn from a standard normal distribution. The discriminator $D(\mathbf{x};\theta_d)$ is also a feedforward neural network with parameters $\theta_d$ which maps an element $\mathbf{x}$ of the data space to a scalar in the interval $[0,1]$. The output of $D$ is the predicted probability that the input is an element from the training data rather than a counterfeit generated by $G$. $D$ is trained to maximize the probability of assigning a correct label to training data and samples from $G$, whereas $G$ is trained to minimize the probability of $D$ correctly marking its samples as counterfeit. That is, $D$ and $G$ play a zero-sum minimax game with the value function $V(D,G)$ (the value function is the value obtained by the loss function at equilibrium):

$$\min_G \max_D V(D,G) = \mathbb{E}_{\mathbf{x}}[\log D(\mathbf{x})] + \mathbb{E}_{\mathbf{z}}[\log(1 - D(G(\mathbf{z})))] \tag{10}$$

Although elegant, it is more practical to train $G$ to maximize $\log(D(G(\mathbf{z})))$ rather than minimize $\log(1 - D(G(\mathbf{z})))$. This is because, with the latter loss function, early in training when $G$ is ineffective, $D$ rejects samples from $G$ with high confidence. In this case, the gradients for $G$ are small and training $G$ is slow.

## III Data Preparation

Rather than working with raw images from the PICO-60 cameras, it is desirable to apply some pre-processing to remove background and noise before applying machine learning methods. This will reduce complexity in the images, accentuate bubbles, and simplify generation of artificial images. To do this, we implement an image processing algorithm devised by Pitam Mitra in Ref. [13]. This algorithm removes background from the images to leave only the bubbles. The resulting image is binarized such that the pixels belonging to each bubble are white and all others are black.

First, fiducial and multiplicity cuts are applied to the data to isolate single-bubble events within the fiducial bulk volume. The fiducial cut removes bubbles within 6 mm of the jar walls and freon-water boundary. For regions where optics are worse such as the transition from the cylinder to the lower hemisphere, bubbles within 13 mm of the wall are removed [14]. These cuts are done to remove events occurring as a result of interactions of the target liquid with the walls or water buffer, which are not of interest.

Image analysis algorithms from the OpenCV [15] computer vision package are used for image manipulation

and processing in Python [16]. We then loop over all events passing the cuts and process the images as follows:

1. The first two images from every event in the current run are used as the training dataset. The mean $\mu_T$ and standard deviation from the mean $\sigma_T$ of each pixel in the training images are computed. An example of these images is shown in Fig. 3.

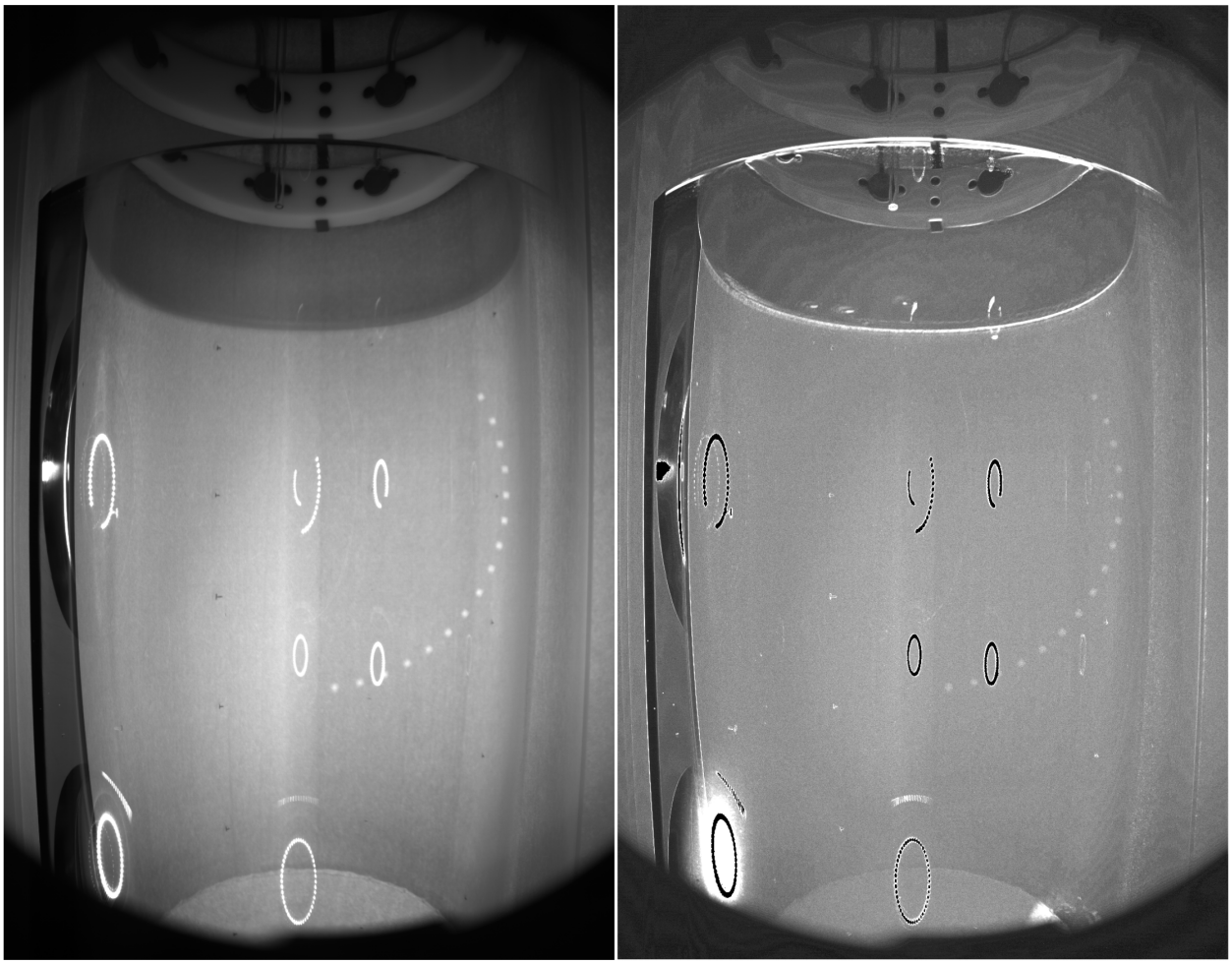

Figure 3: Mean (left) and standard deviation (right) of each pixel from the first two frames of all 16 events in run 20170214_3, camera 2. The dynamic range of the standard deviation was scaled to aid in visualization.

2. The trigger frame $F$ (first frame containing a bubble) is loaded and the absolute difference with the mean is taken: $F_\mu = |F - \mu_T|$. The trigger frame was previously found using the AutoBub algorithm, an algorithm for detection and tracking of nucleation bubbles described in Ref. [13]. An example of an image $F_\mu$ is shown in Fig. 4.

3. The image $F_{6\sigma} = F_\mu - 6\sigma_T$ is then computed. This represents pixels which are at least $6\sigma$ above the noise. An example of an image $F_{6\sigma}$ is shown in Fig. 4.

4. A median blur with a $3 \times 3$ kernel is then applied to $F_{6\sigma}$. The median blur runs over every pixel in the image, replacing its value with the median of its neighbouring entries.

5. A threshold with pixel intensity 3 is applied. Any pixel that does not pass the cut is set to zero. All pixels passing the cut retain their original value.

Figure 4: $F_\mu$ (left) and $F_{6\sigma}$ (right) from the processing of a single-bubble event from run 20170214_3, event 8, camera 2. The colour has been inverted and pixel intensity has been scaled to aid in visualization.

6. Thresholding is applied again, but this time with a threshold determined by Otsu's binarization [17]. Otsu's binarization method is described in further detail in Appendix A. Pixels which pass this threshold should belong to bubbles and are set to white. All other pixels are set to black.

An example of an image before and after processing is shown in Fig. 5.

The pixel coordinates of the bubbles were previously computed using the AutoBub algorithm. The processed images are cropped to 32 pixel square images centered on each bubble. These binary bubble images then are saved. An example of a few bubbles is shown in Fig. 6. In total, 1134 single-bubble events were processed to obtain 3297 bubble images. A datafile containing the run ID, event number, 3D position within the detector, and pixel coordinates for all bubbles is saved for future reference.

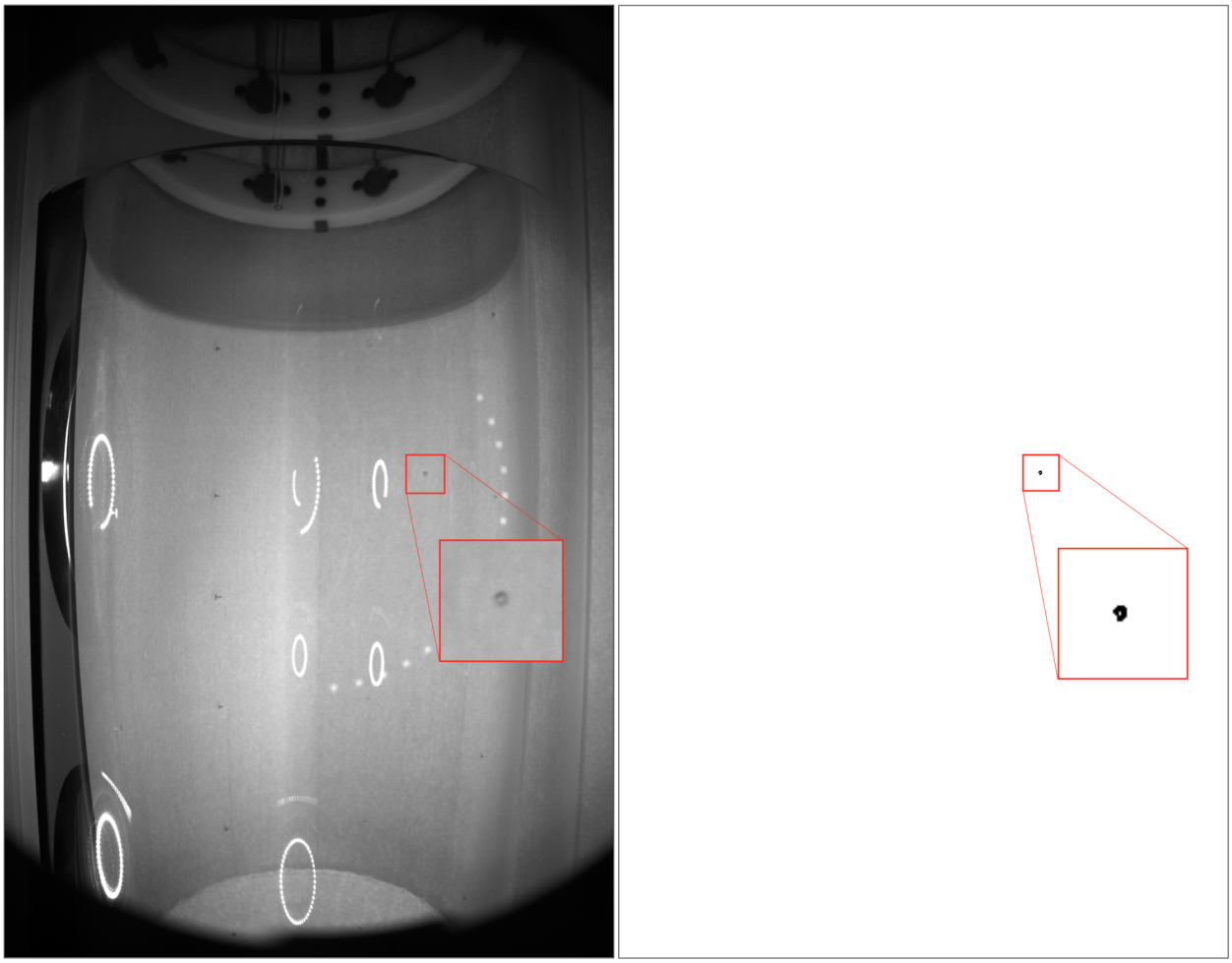

Figure 5: Trigger frame of a single bubble event before and after processing with the bubble magnified. The colour has been inverted in the processed image to aid in visualization. This is from run 20170214_3, event 8, camera 2.

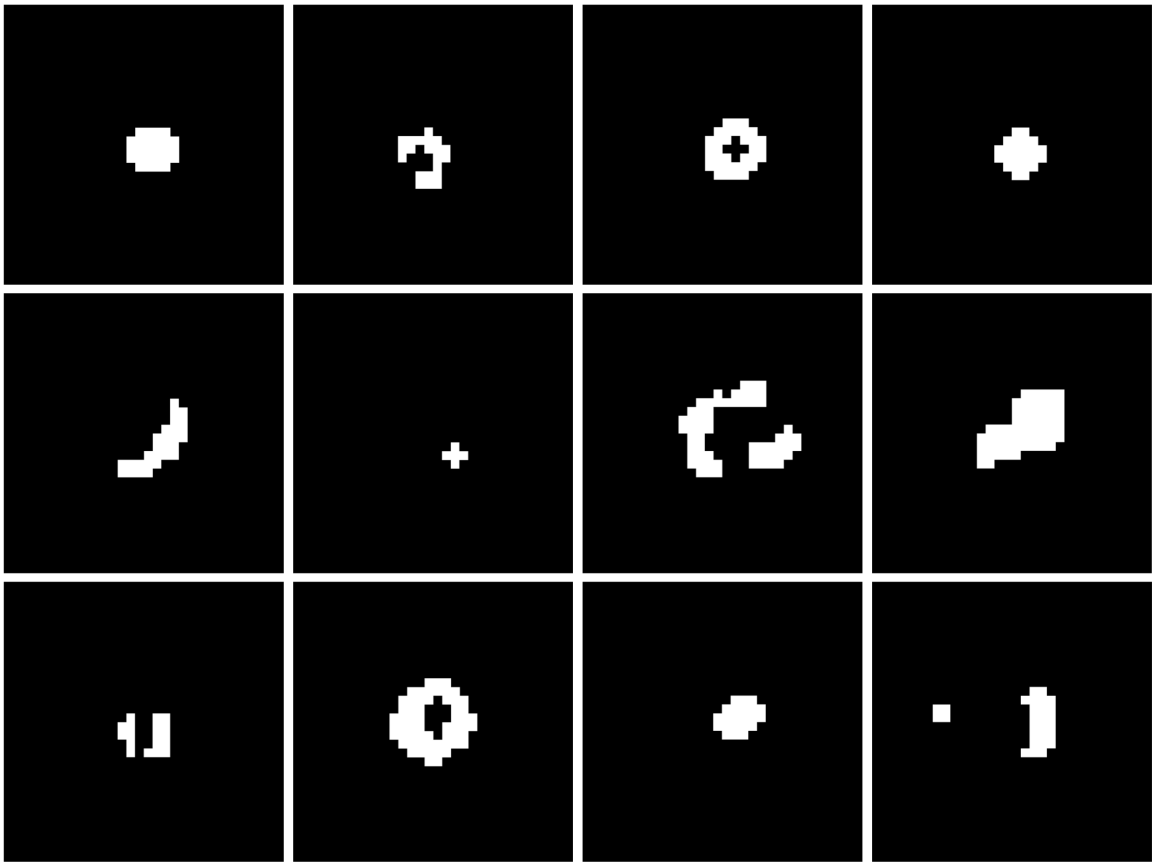

Figure 6: Samples of 32 pixel square bubble images obtained through processing single bubble events.

# IV Generating Artificial Bubbles

We now wish to apply a generative adversarial network to generate artificial 32 pixel square bubble images, modeling the distribution of images in the processed image dataset. To do this, we employ the open source machine learning library PyTorch [18] for Python. PyTorch enables automatic gradient computations and execution of dynamic tensor analysis while maintaining performance comparable to the fastest current deep learning libraries.

Several variants of the standard GAN architecture have been developed and tested in various applications [19]. For the task of generating bubble images, four GAN architectures were tested: the original GAN architecture [12] with fully-connected layer based networks (named GAN-Lin) and with CNNs (named GAN-Conv); boundary equilibrium GAN (BEGAN) [20]; and deep convolutional GAN (DCGAN) [21]. BEGAN replaces the standard loss computation with one based on the Wasserstein distance, a metric for probability distributions on a metric space. DCGAN implements a set of constraints on the architecture of the generator and discriminator networks, which are both CNNs.

Unlike standard neural network models which are trained with a loss function until convergence, a GAN generator's loss depends on the performance of the discriminator and vice versa. This makes quantitative comparison of GAN models a difficult task. Although several evaluation measures for GANs exist [22], they often require significant groundwork to implement, and may not capture all of the aspects of the image datasets we wish to compare. For this reason, the models were evaluated based on qualitative observations only. Sample images from each of the models are shown in Fig. 7.

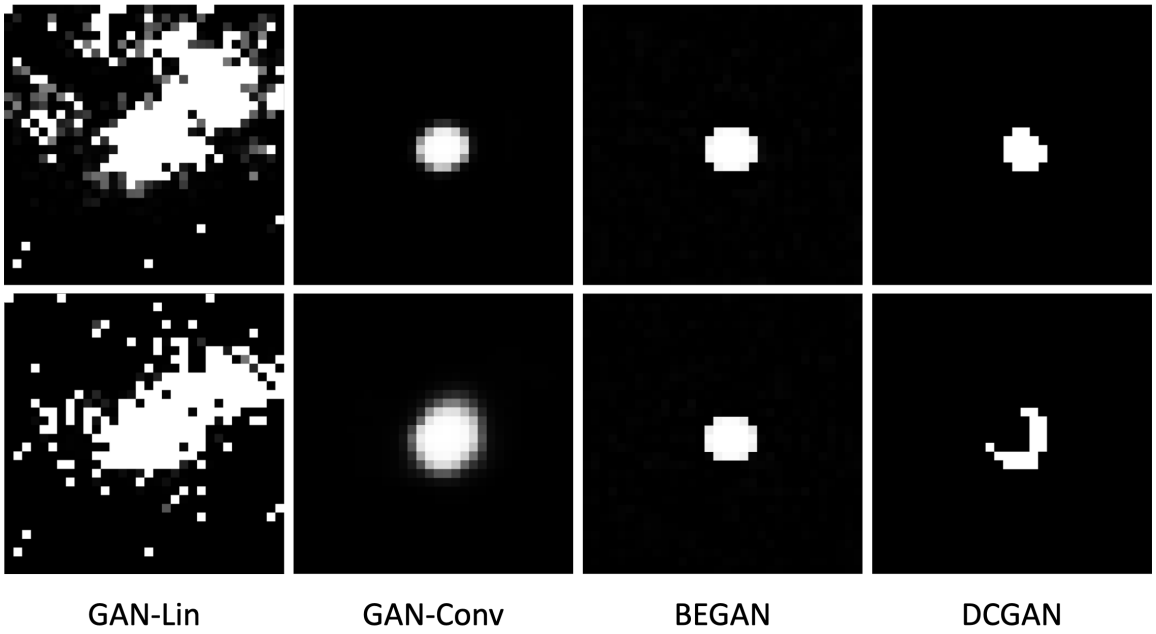


Figure 7: Samples of 32 pixel square artificial bubble images obtained from each of the GAN architectures tested.

The BEGAN network demonstrated fast model generation, but the model lacked in image diversity. Images generated by this model appeared as a single, round bubble in the center of the image, and did not capture the variety of bubble shapes seen in the data. In the same manner, the model generated by GAN-Conv lacked image diversity. However, this architecture was much slower to generate a satisfactory model than BEGAN. As seen by the images in Fig. 7, GAN-Lin was not able to produce bubble images with

any believable resemblance to the data. DCGAN was the slowest to generate a model, but produced high quality images. The artificial bubbles had a strong visual resemblance to the data, and, alike to the data, demonstrated a large variety of bubble shapes and sizes.

The DCGAN model was trained with latent space parameters drawn from a standard normal distribution, but upon generating artificial bubbles, it was found that adjusting the standard deviation of the distribution affected the complexity of the bubble images generated. This effect is demonstrated in Fig. 8, and is a well-known property of latent space variables in GANs [23, 24].

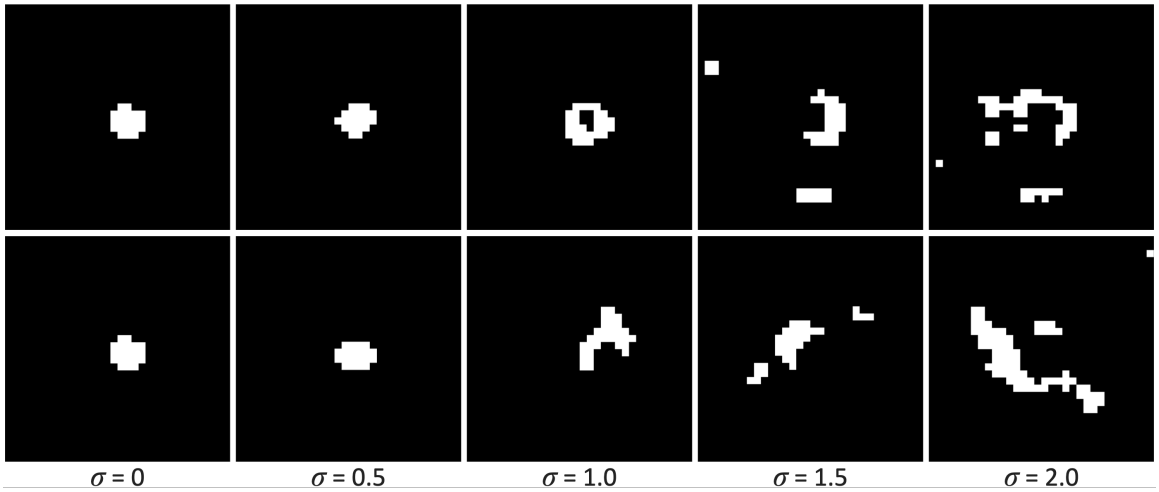


Figure 8: Artificial bubble images generated by DCGAN for an increasing standard deviation $\sigma$ of the latent space parameters. For $\sigma = 0$, the input is always a zero vector, so every bubble generated appears the same as the ones shown.

By qualitative observation, it was found that reducing the standard deviation of the distribution to around 0.85 was most effective in generating realistic counterfeits.

# V Ray Tracing

Ray tracing is a rendering technique for generating images by propagating rays of light through a virtual geometry and simulating the effects of interactions with virtual objects [25]. This is done by simulating the process of image capture in a camera in reverse. An image in a camera is produced when light reflected off objects in the environment is absorbed by pixels in the camera. In ray tracing simulations, we instead begin with a ray at each pixel and propagate the rays into the environment. This allows us to determine the objects with which the light interacted prior to entering the camera and the pixel coordinates corresponding to the object in the image.

MATLAB [26] ray tracing code, originally written by Eric Dahl and later modified by Gavin Crowder and Clarke Hardy was modified and used to project rays from the PICO-60 cameras into the detector geometry. The code was adapted to compute trajectories of all rays and the pixel coordinates corresponding to each ray. We begin rays at each of the pixels in each camera and propagate them through the detector geometry, terminating propagation when they escape the geometry. Reflected rays are not tracked as they are not of interest. An array containing the starting point and scattering points for each ray, the pixel coordinates and camera corresponding to each ray, and a few geometric parameters necessary for future steps was computed and returned. To incorporate this code into the remaining analysis in Python, the MATLAB Engine API

for Python is used. This allows MATLAB functions to be run in a Python environment.

To generate artificial images, we first need to choose a position in 3D space within the detector volume for the artificial bubble to be placed. A bounding box surrounding the detector volume is defined, and points are randomly sampled uniformly from within the box. We then check if the point is within the target volume. If it is, we keep it. Otherwise, a new point is sampled and checked. It is straightforward to check if the point lies within the cylindrical portion of the jar (region III in Fig. 9) or the spherical portion of the jar dome (region I in Fig. 9). Determining if the point is within the section of the jar bounded by the torus-shaped knuckle (region II in Fig. 9) requires further computation. This computation is executed in Appendix B.

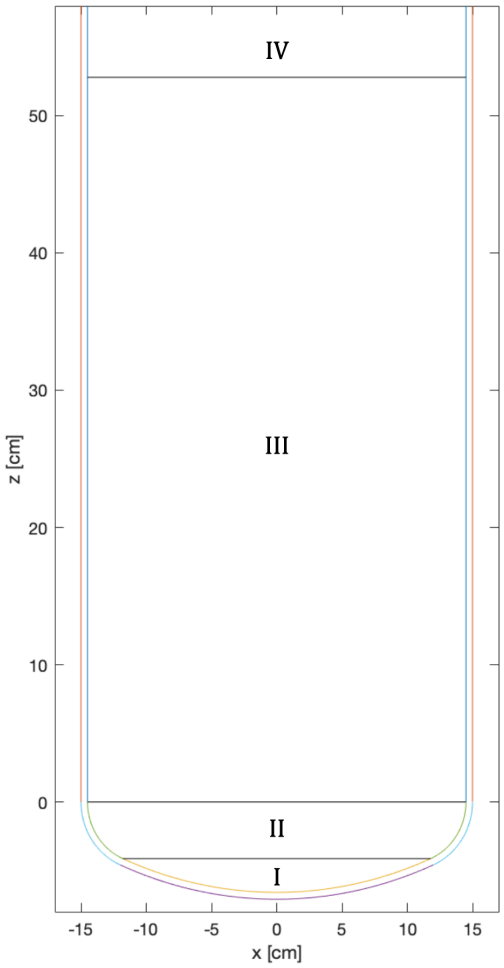


Figure 9: Cross section of the silica jar with regions separated by horizontal black lines. In region I, the inner surface of the jar is defined by a sphere of radius 29.5 cm centered at $(0, 0, 22.9$ cm). In region II, the inner surface of the jar is defined by a torus in the $x$-$y$ plane centered at $(0, 0, 0)$ with major radius 10.0 cm and minor radius 4.5 cm. The inner surface of the jar in regions III and IV is defined by a cylinder coaxial with the $z$-axis of radius 14.5 cm. The boundary between regions III and IV gives the interface between the water buffer and the freon.

After sampling a random point $\mathbf{p}$ in the detector volume, we compute the pixel coordinates of $\mathbf{p}$ in each of the cameras. To do this, we first compute the minimum Euclidean distance of $\mathbf{p}$ to each of the rays. Each ray through the detector geometry is a line defined by starting and ending points $\mathbf{x}_0$ and $\mathbf{x}_1$ respectively.

The minimum distance from $\mathbf{p}$ to the ray is then

$$d = \frac{||(\mathbf{x}_0 - \mathbf{p}) \times (\mathbf{x}_1 - \mathbf{p})||}{||\mathbf{x}_1 - \mathbf{x}_0||}. \tag{11}$$

For each camera, the ray with the minimum distance to $\mathbf{p}$ is selected. The pixel coordinates associated with the selected rays are then obtained from previously saved data. If $d$ is larger than an empirically determined threshold for the selected ray, $\mathbf{p}$ is considered to not be visible in the current camera.

Computing and minimizing $d$ for every ray was found to be quite computationally expensive. To mitigate this issue, we partition the bounding box surrounding the detector volume into 3cm×3cm×3cm cubes. For each cube, the rays passing through that cube are determined and saved in a pickle file. Then, given $\mathbf{p}$, we are only required to minimize $d$ over all rays passing through the cube containing $\mathbf{p}$, reducing computation times drastically.

# VI Image Generation

Bubble radius, measured in pixels as seen in each camera, varies based on the distance of the bubble from the camera, trigger timing, and other exogenous factors. For this application, we assume bubble radius varies linearly with distance from the camera. To determine bubble radius, we employ OpenCV's `cv2.findContours` method. This method computes and returns contours found in a binary image. We invoke the option `cv2.RETR_EXTERNAL` to compute only the external most contours and ignore "holes" in bubbles. The moments of the contour up to the third order are then computed via the OpenCV method `cv2.moments`. For a contour $C$, spatial moments are computed as

$$M_{ij} = \sum_{x,y} C(x,y) x^i y^j \tag{12}$$

and can be used to compute several contour properties including area and center of mass. The center of the bubble is defined to be its center of mass and is computed as

$$(\bar{x}, \bar{y}) = \left( \frac{M_{10}}{M_{00}}, \frac{M_{01}}{M_{00}} \right). \tag{13}$$

The bubble radius is then defined to be the median distance of the contour from its center of mass. For each camera, the coefficients of the (assumed) linear relation between bubble radius and Euclidean distance from the camera for the PICO-60 data were obtained through simple linear regression.

The open source scientific computing package NumPy [27] is used to create and manipulate arrays representing the artificial images, and reading and writing images is done with OpenCV. Artificial images are generated as follows:

1. First, we initialize four blank images. These are 1088×1700 NumPy arrays filled with zeros, and bubble images will be embedded into these arrays to create the artificial images.

2. Next, we sample a random point $\mathbf{p}$ within the target volume of the detector and compute pixel coordinates in each of the cameras via the method described in Section V.

3. Four random bubbles from the dataset of artificial bubble images previously created by DCGAN are then sampled. One bubble is assigned to each camera.

4. The bubble images are then scaled based on the relative position of $\mathbf{p}$ from each of the cameras. To accomplish this, the radius $r_{\mathbf{p}}$ of a bubble at $\mathbf{p}$ is computed based on the linear relation determined previously from the PICO-60 data. The radius $r$ of the artificial bubble is then computed, and the artificial bubble image is scaled by $r_{\mathbf{p}}/r$.

5. The scaled bubble images are then inserted into the blank detector images at the computed pixel coordinates.

6. Steps 2. to 5. are then repeated until the desired multiplicity of the artificial event has been reached.

7. Finally, the four completed artificial images are saved along with a datafile listing the pixel coordinates and 3D position of each bubble.

The result is a set of four binary images corresponding to each of the cameras. An example of a 5-bubble artificial event is shown in Fig. 10.

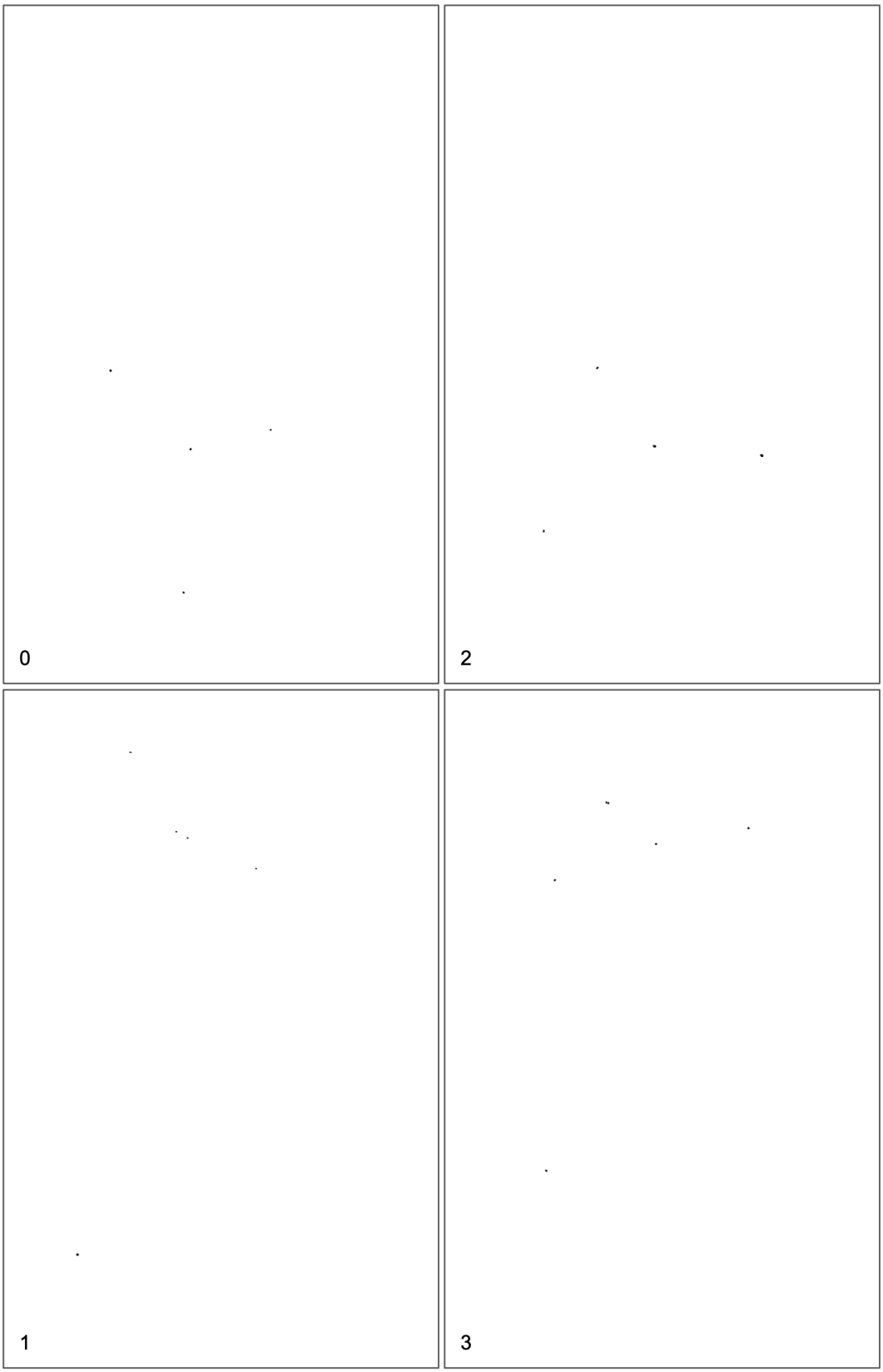


Figure 10: Artificial images from a 5-bubble event. Camera number is written in the bottom left of each image. Color has been inverted to aid in visualization.

# VII Image Analysis

50000 single bubble artificial events were generated and compared to the PICO-60 data. The bubble pixel position distributions in both the PICO-60 images and the artificial images were plotted in Fig. 11. As much of the PICO-60 data arises from calibration runs using a radioactive source, the pixel coordinates were weighted by the square of their distance from the source to correct for the source positioning.

The Pearson correlation coefficient $r$ was used as a similarity metric to compare histograms [28, 29]. $r$ is shown for each camera in Fig. 11, and was computed after adjusting the artificial image histogram bin width to 90 pixels to match the histograms created from the PICO-60 data. These correlation coefficients indicate a strong correlation between artificial and real bubble distributions in the images. A strong visual similarity is also present.

The increase in bubble density near the right (for cameras 0 and 2) or left (for cameras 1 and 3) of the position distributions is likely due to the pitch of the cameras. This causes the jar to appear to narrow at one end, increasing the bubble density in that region as seen by the cameras.

Radius of bubbles, as computed in Section VI, was plotted against the Euclidean distance of the bubble from the camera for the PICO-60 data and the artificial images, and is shown in Fig. 12. To avoid optical distortion effects from the curvature of the jar, only bubbles with y-coordinate between 394 and 694 were considered. Additionally, for cameras 1 and 3, we require the x-coordinate of the bubble to be less than 1250.

Assuming the radius is a linear function of distance from the camera, the fit parameters of the linear regression agree within uncertainty between the PICO-60 and artificial images. For the purposes of artificial image generation, this indicates the model demonstrates a sufficient radius estimation and scaling procedure for artificial bubble images.

The outlying bubbles in the PICO-60 data with significantly larger radius than the majority of bubbles are likely caused by late triggers. If the camera system fails to recognize an event, or triggers recompression significantly after bubbles form, the bubbles have time to grow to a large radius before they are imaged.

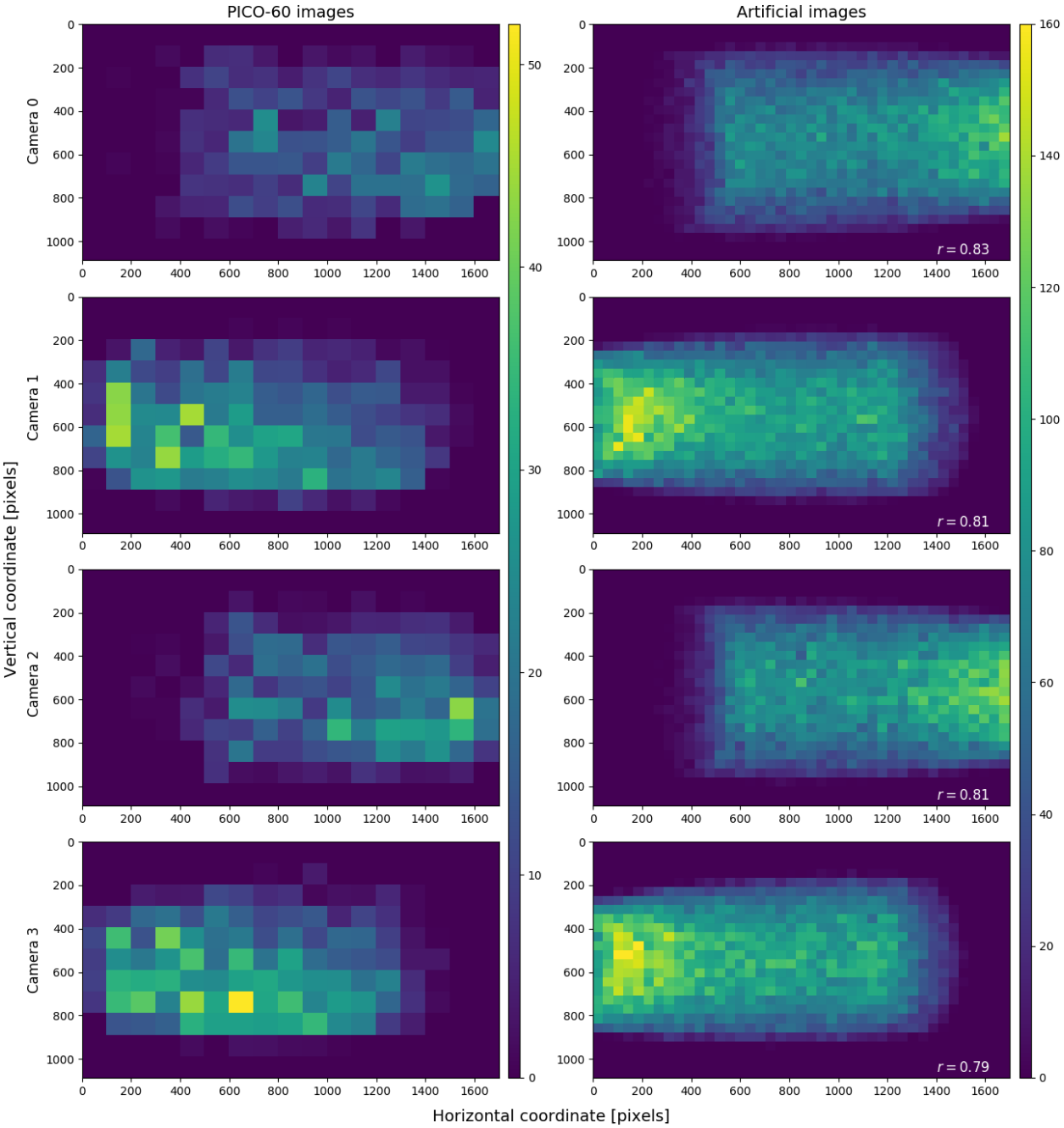


Figure 11: Distribution of pixel coordinates of bubbles in each of the four cameras. PICO-60 images appear in the left column and are based on 1994 events with square bins of width 90 pixels. Artificial images appear in the right column and are based on 50000 events with square bins of width 40 pixels. The coordinates in the events in the PICO data which arise from calibration data are weighted by their squared distance from the calibration source to correct for source position. The Pearson correlation coefficient $r$ between histograms is shown for each camera.

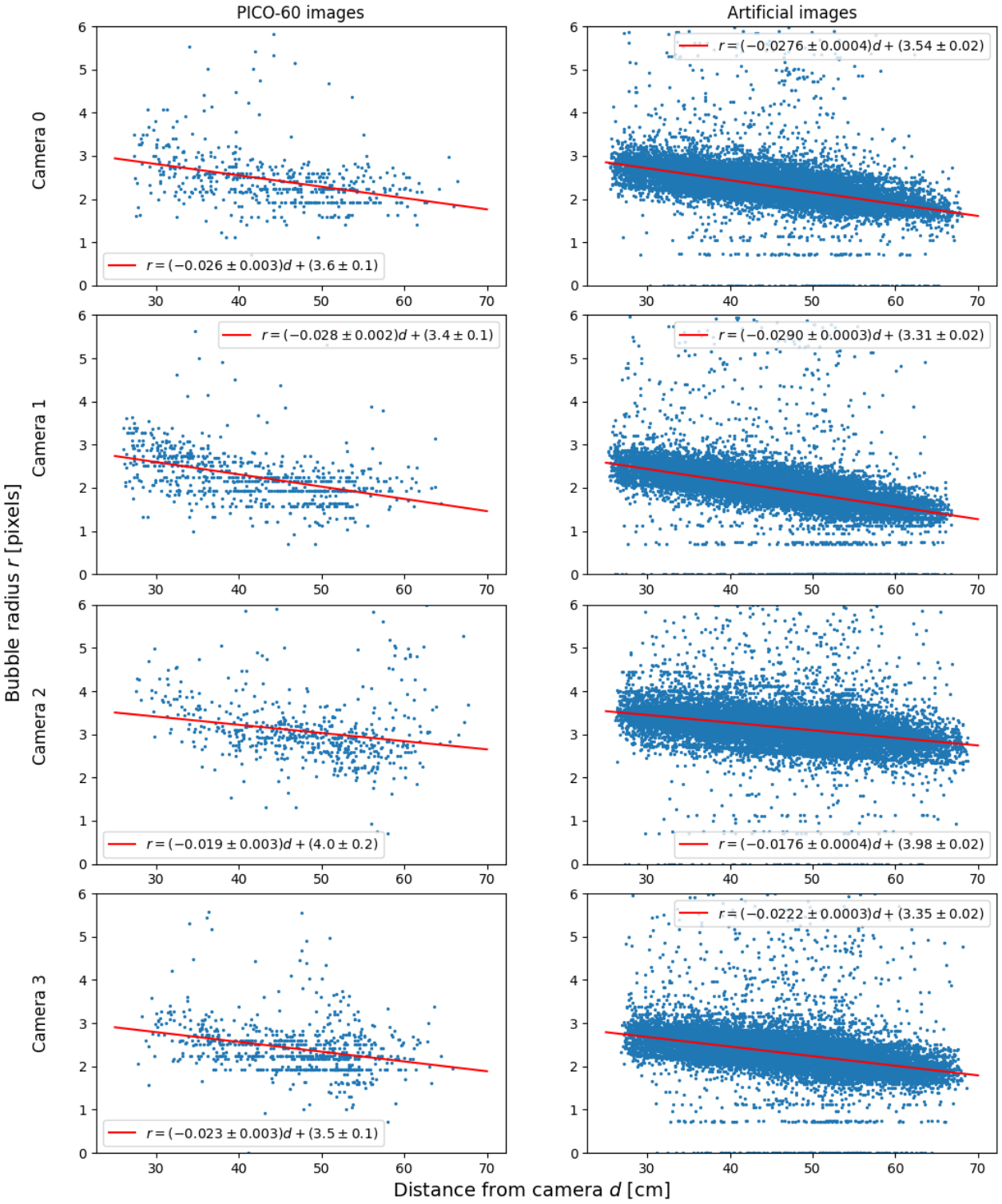


Figure 12: Pixel radius of bubbles vs. Euclidean distance between the bubble and camera with a linear fit. Only bubbles with y-coordinate between 394 and 694 are considered. For cameras 1 and 3, we additionally require the bubbles x-coordinate to be less than 1250. This is done to exclude bubbles near the jar walls which are distorted due to the curvature of the jar.

# VIII Future Work

Although likely sufficient for transfer learning applications, improvements can be made to the artificial images in the following ways:

- Bubble positions were drawn uniformly from within the detector volume before computing pixel coordinates. However, the relative positions between bubbles in multiple-bubble events are not uniformly distributed as they are caused by multiple scattering of neutrons. This could be amended by preferentially selecting bubbles near each other, or by simulating multiple scattering events using Monte Carlo simulation techniques or other methods.

- Four bubble images corresponding to the same bubble were drawn randomly from the artificial bubble dataset when generating the artificial event images. This may cause the same bubble to look vastly different in each of the four cameras. To counter this, bubbles of a similar shape could be selected before embedding into the event images. Some variation in bubble shape is seen between cameras in the PICO-60 dataset, so it is not essential to match bubble shapes exactly.

- Optical distortion of bubbles due to the curvature of the jar was not considered. The ray tracing code mentioned in Section V could be utilized to characterize this distortion at points throughout the jar, and bubble images could be deformed to match this distortion before composing event images.

An effective 3D position reconstruction algorithm is important for event selection and rejection based on topological properties. This enables rejection of neutron multiple scattering events as well as events with bubbles occurring near the jar walls or freon-water interface which arise due to interactions of the target liquid with the bounding material. Transfer learning [6] techniques allow neural networks trained on data from a given distribution to benefit from networks trained on data from similar distributions. Using the artificial data, a neural network for 3D position reconstruction could be developed. The abundance of available training examples should facilitate generation of an accurate and precise model. As the data distributions corresponding to artificial and PICO-60 images should be adequately similar, this network could be directly applied to 3D position reconstruction of PICO-60 events.

---


[1] C. Amole *et. al.*, Dark matter search results from the PICO-60 CF3I bubble chamber, *Physical Review D*, 93, 052014 (2016). doi: 10.1103/PhysRevD.93.052014

[2] C. Amole *et. al.*, Dark matter search results from the complete exposure of the PICO-60 C3F8 bubble chamber, *Physical Review D*, 100, 022001 (2019). doi: 10.1103/PhysRevD.100.022001

[3] X. Han, H. Laga, and M. Bennamoun, Image-based 3D Object Reconstruction: State-of-the-Art and Trends in the Deep Learning Era, *IEEE Transactions on Pattern Analysis and Machine Intelligence* (2019). doi: 10.1109/TPAMI.2019.2954885

[4] Z. Zhao, P. Zheng, S. Xu, and X. Wu, Object Detection With Deep Learning: A Review, *IEEE Transactions on Neural Networks and Learning Systems*, 30(11):3212-3232 (2019). doi: 10.1109/TNNLS.2018.2876865.

[5] S. Femg, H. Zhou, and H. Dong, Using deep neural network with small dataset to predict material defects, *Materials & Design*, 162:300-310 (2019). doi: 10.1016/j.matdes.2018.11.060.

[6] Y. Bengio *et. al.*, Deep learners benefit more from out-of-distribution examples, *JMLR: W&CP: Proc. AISTATS*, 15:164-172 (2011).

[7] J.C. Spall, Introduction to Stochastic Search and Optimization: Estimation, Simulation, and Control, *John Wiley & Sons*, Hoboken, New Jersey (2003).

[8] D. Kingma and J. Ba, Adam: A Method for Stochastic Optimization, *arXiv:1412.6980* [cs.LG] (2014).

[9] F. Yu, V. Koltun, Multi-Scale Context Aggregation by Dilated Convolutions, *arXiv:1511.07122* [cs.CV] (2015).

[10] G. Hinton *et. al.*, Deep neural networks for acoustic modeling in speech recognition, *IEEE Signal Processing Magazine*, 29(6):82–97 (2012). doi: 10.1109/MSP.2012.2205597

[11] A. Krizhevsky, I. Sutskever, and G. Hinton, ImageNet classification with deep convolutional neural networks, *Communications of the ACM*, 60(6):84-90 (2017). doi: 10.1145/3065386

[12] I.J. Goodfellow *et. al.*, Generative Adversarial Networks, *Proceedings of the 27th International Conference on Neural Information Processing Systems*, 2:2672-2680 (2014).

[13] P. Mitra, PICO-60: A Dark Matter Search Experiment with C3F8 in a Bubble Chamber, *Ph. D. thesis*, University of Alberta (2018).

[14] C. Amole *et. al.*, Dark Matter Search Results from the PICO-60 C3F8 Bubble Chamber, *Physical Review Letters*, 118, 251301 (2017). doi: 10.1103/PhysRevLett.118.251301

[15] G. Bradski, The OpenCV Library, *Dr. Dobb's Journal of Software Tools* (2000).

[16] G. Van Rossum and F.L. Drake, Python 3 Reference Manual, Scotts Valley, CA, *CreateSpace* (2009).

[17] N. Otsu, A Threshold Selection Method from Gray-Level Histograms, *IEEE Transactions on Systems, Man, and Cybernetics*, 9(1):62–66 (1979). doi:10.1109/ TSMC.1979.4310076.

[18] A. Paszke *et. al.*, PyTorch: An Imperative Style, High-Performance Deep Learning Library, *Advances in Neural Information Processing Systems 32* (2019).

[19] E. Linder-Norén, PyTorch-GAN: PyTorch implementations of Generative Adversarial Networks, *GitHub repository*, Retrieved February 2020 from https://github.com/eriklindernoren/PyTorch-GAN (2019).

[20] D. Berthelot, T. Schumm, and L. Metz, BEGAN: Boundary Equilibrium Generative Adversarial Networks, *arXiv:1703.10717* [cs.LG] (2017).

[21] A. Radford, L. Metz, and S. Chintala, Unsupervised Representation Learning with Deep Convolutional Generative Adversarial Networks, *arXiv:1511.06434* [cs.LG] (2015).

[22] A. Borji, Pros and cons of GAN evaluation measures, *Computer Vision and Image Understanding*, 179:41-65 (2019). doi: 10.1016/j.cviu.2018.10.009

[23] A. Plumerault, H. Le Borgne, and C. Hudelot, Controlling generative models with continuous factors of variations, *International Conference on Learning Representations* (2020).

[24] A. Jahanian, L. Chai, and P. Isola, On the "steerability" of generative adversarial networks, *International Conference on Learning Representations* (2020).

[25] C. Hardy, Tech Note: MATLAB Ray Tracing Code, Retrieved March 2020 from https://coupp-docdb.fnal.gov (2020).

[26] MATLAB:2017b, *The MathWorks, Inc.*, Natick, Massachusetts, United States.

[27] T.E. Oliphant, A guide to NumPy, Vol. 1, *Trelgol Publishing USA* (2006).

[28] K. Meshgi and S. Ishii, Expanding Histogram of Colors with Gridding to Improve Tracking Accuracy, *14th IAPR International Conference on Machine Vision Applications*, 475-479 (2015). doi: 10.1109/MVA.2015.7153234

[29] K.H. Liland, T. Naes, and U.G. Indahl, MatrixCorrelation: Matrix Correlation Coefficients, *R Core Team* (2017).

## Appendix A: Otsu's Binarization

Derived in 1979 by Nobuyuki Otsu, Otsu's binarization [17] is a method used to perform automatic image thresholding to binarize a grayscale image. First, a histogram of pixel intensities is computed. Each threshold $T$ splits the $L$ bins of the histogram into two classes: those passing the threshold (class 1) and those not passing (class 0). The intraclass variance is defined as the weighted sum

$$\sigma_w^2(T) = w_0(T)\sigma_0^2(T) + w_1(T)\sigma_1^2(T) \tag{14}$$

where $\sigma_0^2$ and $\sigma_1^2$ are the variances of the two classes and $w_i(T)$ is the class probability computed as

$$\begin{aligned} w_0(T) &= \frac{\sum_{i=1}^{T} h(i)}{\sum_{i=1}^{L} h(i)} \\ w_1(T) &= \frac{\sum_{i=T+1}^{L} h(i)}{\sum_{i=1}^{L} h(i)} \end{aligned} \tag{15}$$

where $h(i)$ gives the count of the histogram bin corresponding to pixel value $i$. Otsu's binarization method exhaustively searches for the threshold that minimizes the intraclass variance. This method exhibits relatively good performance if the image can be assumed to have bimodal distribution and possess a deep valley between two sharp peaks [30].

## Appendix B: Points Within a Torus

Here, we wish to determine if a randomly selected point $\mathbf{p} \in \mathbb{R}^3$ is within a toroidal domain. We consider a torus in the $x$-$y$ plane with major radius $a$ and minor radius $b$ centered at $(0, 0, z_0)$. Given $\mathbf{p} = (x, y, z) \in \mathbb{R}^3$, we compute the intersection of the circle of radius $a$ at the center of the torus and the plane containing the $z$-axis and $\mathbf{p}$. There are two such intersection points, so we chose the one $\tilde{\mathbf{p}}$ with $x$-$y$ coordinates in the same quadrant as $\mathbf{p}$. This point is given by

$$\begin{aligned} \tilde{\mathbf{p}} &= \mathcal{P}\left(\frac{a\mathbf{p}}{||\mathbf{p} - (0, 0, z)||}\right) \\ &= \left(\frac{ax}{\sqrt{x^2 + y^2}}, \frac{ay}{\sqrt{x^2 + y^2}}, z_0\right) \end{aligned} \tag{16}$$

where $\mathcal{P}$ is the projection onto the plane of the torus. Those points within the torus then satisfy

$$||\tilde{\mathbf{p}} - \mathbf{p}|| < b. \tag{17}$$

If we also want to accept points within the "hole" of the torus, we accept points within the cylinder of radius $a$ and height $2b$ that is concentric and coaxial with the torus.

---

[30] J. Kittler and J. Illingworth, On threshold selection using clustering criteria, *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-15(5):652-655 (1985). doi: 10.1109/TSMC.1985.6313443