Tech Note: PICO-60 Artificial Image Generation Code

Adam Smith

March 23, 2020

1 Introduction

This document provides an overview of the usage of the image generation code developed for generation of artificial images with the same statistics as those obtained from the PICO-60 detector. The code uses a generative adversarial network (GAN) to create 32×32 pixel images of bubbles, which are then placed in blank detector images at appropriate positions to produce artificial images.

2 Code Overview and Usage

2.1 Preparing data

To create the dataset on which to train the GAN, we first need to implement an algorithm developed by Pitam Mitra described in Ref. [1] to remove the background and noise from the images and isolate the bubbles. This is implemented in `PICO60_image_analysis.py`. The paths to the directory containing the data for all runs, the ROOT file containing the AutoBub output from all runs, and the desired directory in which to save the images are specified near the top of the code. Fiducial and multiplicity cuts are then applied to the data, and the images are processed into binary images with bubbles white and all other pixels black. The images are then cropped onto 32×32 pixel images centered on each bubble which are saved in the specified output directory. The images are named `run-event-cam-n.png` where `run` is the run ID, `event` is the event number, `cam` is the camera number, and `n` is the bubble number in the event ranging from 0 to one less than the multiplicity of the event (`n` is always zero if we consier only single bubble events). Examples of the output images are shown in Fig. 1.

The function `make_file` from `make_data_file.py` is then called to write a text file containing information about each bubble found in each processed event. Each line corresponds to a single event and contains the run ID, event number, pixel coordinates for cameras 0, 1, 2, and 3, and the 3D position of the bubble within the detector volume in that order.

2.2 Generating artificial bubbles

A GAN is type of neural network architecture in which two convolutional neural networks contest against each other to generate images with the same statistics as a training dataset. The file `dcgan.py` uses the open source machine learning library *PyTorch* to implement a deep convolutional GAN to create a model for generation of bubble images. The path to the directory containing the binary 32×32 pixel bubble images as well as several hyperparameters are specified near the top of the file. Once generated, the model is saved as the PyTorch file `dcgan_model.pt`. The program `dcgan_load.py` runs the model to generate images. This program can be run directly from the command line with the following options:

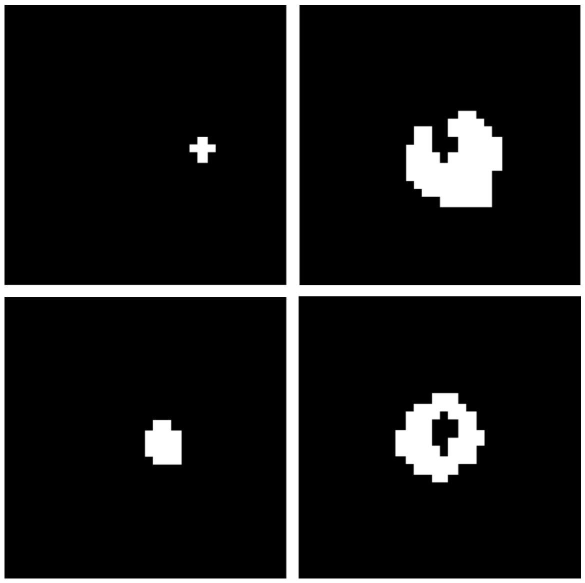

Figure 1: Sample images of bubbles from single bubble events as processed by `PICO60_image_analysis.py`.

- - -*n_images* <int>: Number of images to generate. Default: 100
- - -*stdev* <float>: Standard deviation of Gaussian input data. Default: 0.8
- - -*thresholding* <0 or 1>: Turn on (1) or off (0) thresholding to binarize the image and match the data generated by `PICO60_image_analysis.py`. Default: 1
- - -*threshold* <float>: Threshold in [0,1] to apply if thresholding is on. Default: 0.95

The images are saved in a directory labelled `dcgan_images`.

2.3 Ray tracing

In order to determing pixel coordinates in each of the PICO-60 cameras, we employ the MATLAB ray tracing code originally developed by Eric Dahl. First, we pick a random point within the detector volume for the bubble position, and then use ray tracing to determine the pixel coordinates of the bubble in each of the four cameras. The directory `raytracerfiles` contains all the necessary ray tracing code. Parameters for the detector geometry are specified in `FitPICO60Geometry.m`. This function was adapted from `runthenewgeometryfitter_pico60_2016.m`, and returns an array of parameters defining the PICO-60 geometry. These parameters are used as input for `GetRaysAndPixels.m`, which is the main function carrying out the ray tracing and was adapted from `EricsQuickLookupTableMakerEtcWithTorus.m`. This function traces rays from each of the cameras into the detector geometry. It returns four arrays containing, respectively, the points where each ray scattered, the starting point of each ray, the camera pixel coordinates corresponding to each ray, and some parameters defining the geometry necessary for future steps. Note that this function calls several other ray tracing functions copied directly from previous work which I will not describe here.

2.4 Generating artificial events

The main code for generating artificial events is `generate_event.py`, which is located in the `raytracerfiles` directory. Before this code is run, it is necessary to run `partition_rays.py`. This code runs the MATLAB files described in the previous section, and hence it is necessary to have the MATLAB Engine API for python installed. Pickle files containing the fit parameters generated by `FitPICO60Geometry.m` and the ray information generated by `GetRaysAndPixels.m` are computed and saved for easy and fast future access. A bounding box surrounding the detector geometry is defined, and it is partitioned into 3cm×3cm×3cm boxes. The rays passing through each box for each camera are computed and saved as pickle files in a directory labelled `partitions`.

Artificial events can then be generated by running `generate_event.py`. This code can be run from the command line with the following options:

- - -*n_events* <int>: Number of events to generate. Default: 1
- - -*mult* <int>: Multiplicity of generated events. Default: 1
- - -*image_dir* <str>: Directory in which the 32×32 pixel images of bubbles are saved. Default: `../dcgan_images/`
- - -*write_dir* <str>: Directory in which the artificial events are to be saved. Default: `../artificial_events/`
- - -*separate_events* <0 or 1>: Set to 1 to separate events in different folders, each with their own dictionary and text file. Set to 0 to generate all events in one directory. Default: 0
- - -*omit_txt* <0 or 1>: Set to 1 to omit the text file containing event data. Default: 0
- - -*omit_dict* <0 or 1>: Set to 1 to omit the pickle file containing a dictionary of event data. Default: 0
- - -*omit_imgs* <0 or 1>: Set to 1 to omit the generation of images. Default: 0

To generate an event, a random point within the detector geometry is chosen as the bubble position. The box in the partition containing the bubble position is computed and the corresponding file is read to obtain the rays passing through the box. For each of the four cameras, the distance from the bubble position to each ray is minimized to obtain the ray passing closest to the bubble position. The pixel coordinates corresponding to the selected rays are then computed. Random bubble images are selected and scaled linearly for size based on their distance from each of the cameras. Finally, the scaled bubbles are placed in a blank image at the computed pixel positions to obtain artificial event images. The images are saved and titled `event-cam-mult.png` where `event` is the event number ranging from 0 to one less than the number of events generated, `cam` is the camera number, and `mult` is the multiplicity of the event. An example of the artificial images of a 5 bubble event is shown in Fig. 2.

A text file is created containing one line of data for each bubble. Each line contains the event number, bubble number within the event, 3D bubble position, and pixel coordinates in cameras 0 to 3 in that order. The same information is also saved in a pickle file as a dictionary with the tuple (event number, bubble number) as keys and the tuple (3D bubble position, pixel coordinates) as values for easy access in future python programs. If a bubble is not visible in a specific camera, the pixel coordinates for that camera are recorded as [-1,-1].

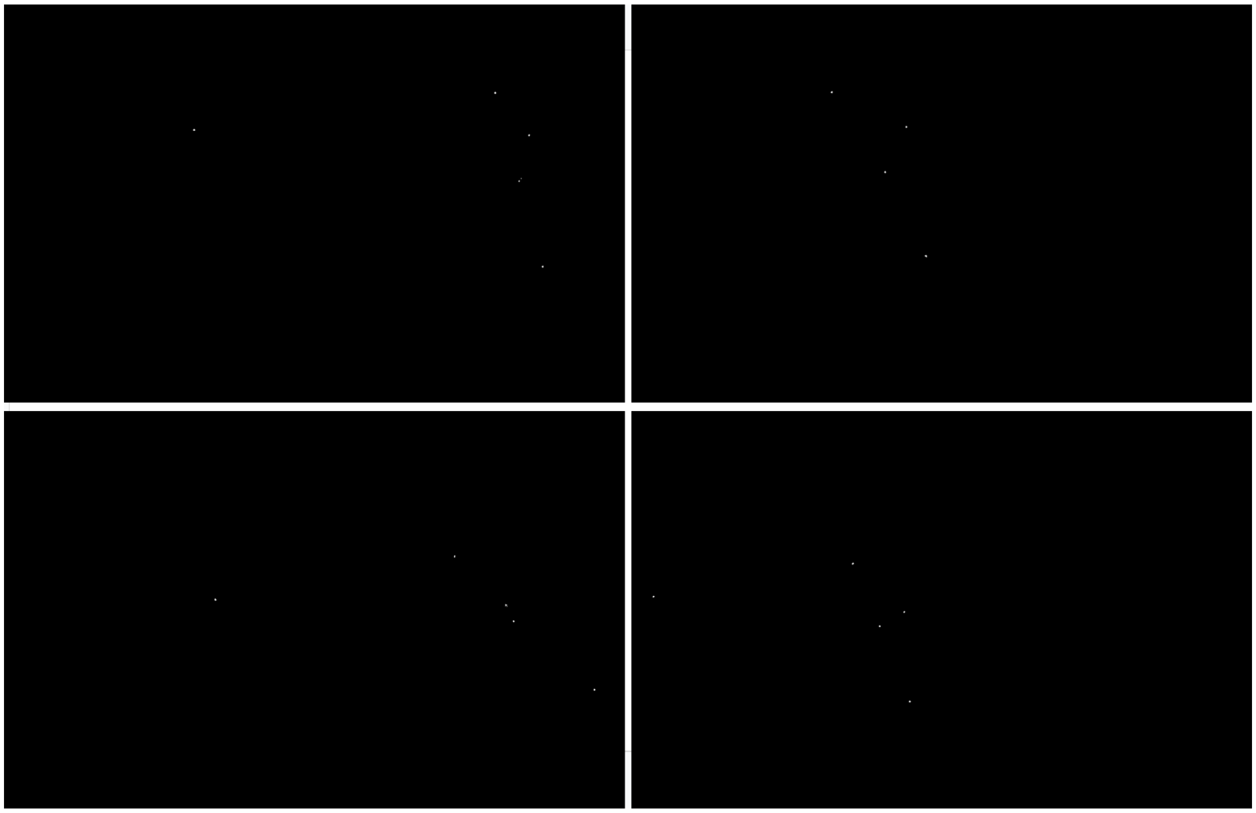

Figure 2: Artificial images of a 5 bubble event. The top left, top right, bottom left, and bottom right images correspond to cameras 0, 1, 2, and 3 respectively.

2.5 Altering geometry

The main files which must be altered to define a new geometry are the MATLAB functions `FitPICO60Geometry.m` and `GetRaysAndPixels.m`. These should be updated to define and propagate rays through the new geometry. The latter should return four arrays containing, respectively, the points where each ray scattered, the starting point of each ray, the camera pixel coordinates corresponding to each ray, and any parameters necessary in future steps.

Moreover, a few items in `partition_rays.py` and `generate_event.py` should be altered. Both programs contain a function `injar` which takes a point in 3D space as input and returns True if the point is within the detector geometry and False otherwise. This function should be the same for both programs and should be changed to suit new geometry accordingly. Also, a few geometry related items were entered manually into these programs, namely

- camera position and resolution;
- the coefficients (found in the function `scale_rad`) of the (assumed) linear realtionship between bubble radius and distance from camera;
- the dimensions and position of a box bounding the entire detector volume.

3 Consistency Checks

For each of the four cameras, the distributions of pixel coordinates were compared for artificial and PICO-60 images and are shown in Fig. 3. This was done to ensure consistency in bubble position and distribution between artificial and real events.

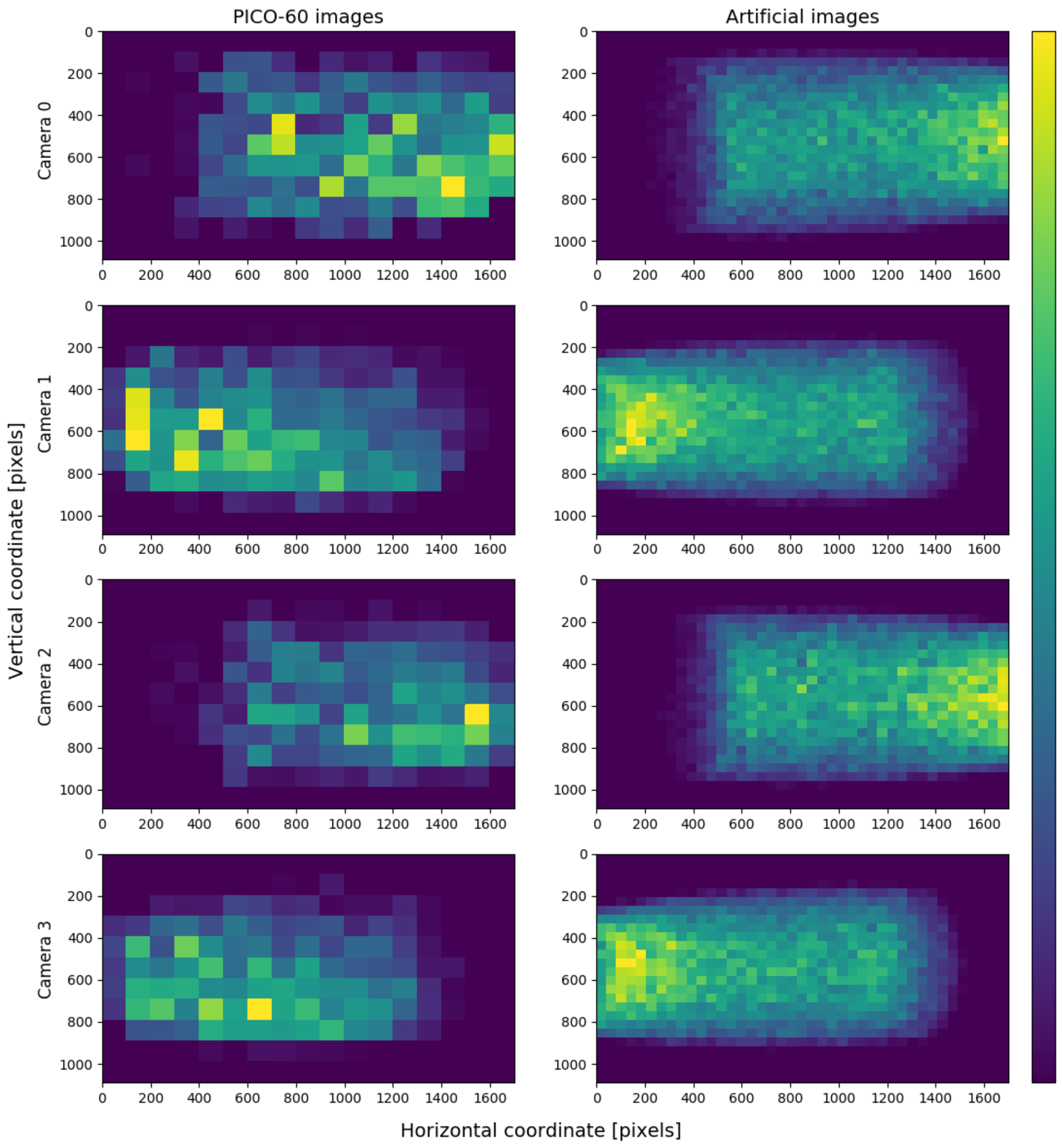

Figure 3: Distribution of pixel coordinates of bubbles in each of the four cameras. The colour scheme follows a linear scale. PICO images are in the left column and are based off 1994 events, and artificial images appear in the right column and are based off 50000 events. The coordinates in the events in the PICO data which arise from calibration data are weighted by their squared distance from the source to correct for source position.

Bubble pixel radius was then compared between the artificial and real datasets, and plots showing the comparison are shown in Fig. 4. The model assumes a linear realtionship between bubble radius and Euclidean distance from the camera, but scales differently for each camera. The linear relationship used in the model was determined empirically based off PICO-60 data.

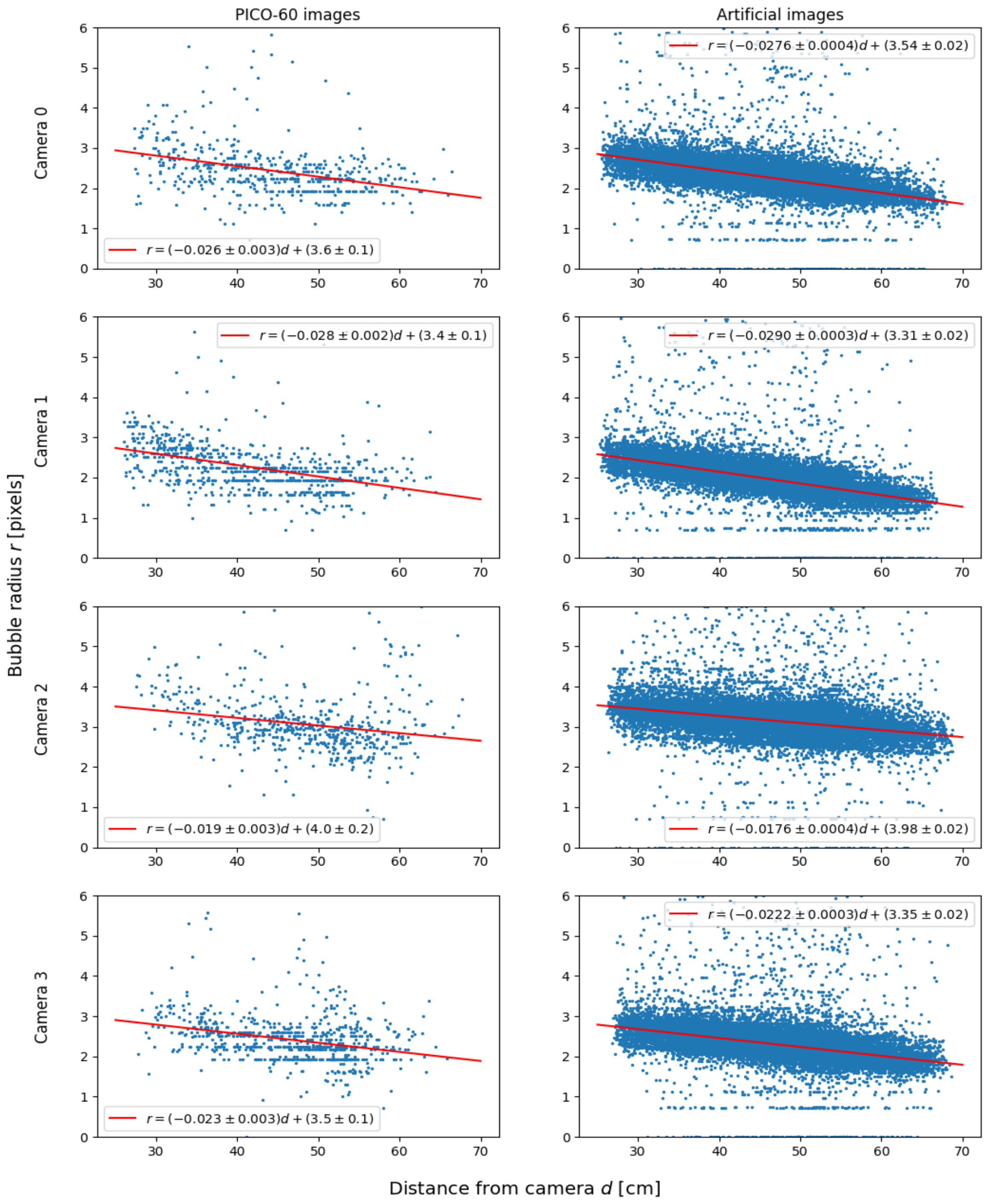

Figure 4: Pixel radius of bubbles vs. Euclidean distance between the bubble and camera. Only bubbles with y-coordinate between 394 and 694 are considered. For cameras 0 and 2, we additionally require the bubbles x-coordinate to be less than 1250. This is done to exclude bubbles near the jar walls which are greatly distorted due to the curvature of the jar.

[1] P. Mitra, Ph. D. thesis, University of Alberta (2018).