KENNESAW STATE
UNIVERSITY

GoMap.py Application: using CARLA Monocular Depth Estimation Algorithm

Acey Dufresne

KSU Identification: 000960649

GoMap.py Application: using CARLA monocular depth estimation algorithm

Acey Dufresne

Abstract:

The goal is to create an algorithm that can understand its surroundings in a three-dimensional manner; in this way the algorithm can be implemented in multiple case settings and appliances. The proposal discusses how the model can be built, and how it can be implemented in the real world.

By creating a dataset in a three-dimensional environment a GAN, generated adversarial network, can be implemented that can take single dimensional input, image-based input, and generate a three-dimensional understanding of this input. The model will deconstruct the image input in composite layers, and re-map these inputs in R^3, or an .fbx format output file. The first portion of this model will be a multi-class classifier, to physically identify the multiple layers of the input. Then the GAN model will make predictions based on each of these classified objects independently, before a generative encoder is used to re-build the image in a three-dimensional environment. This will all be packaged into a UI application, with an additional plugin for Blender.

1. Dataset Preparation:

1.2 Classes and Labels:

1.2.2 Human:

The software *makeHuman* is utilized to populate the dataset with randomly generated three-dimensional objects, specifically human objects. A subscript is initially used, 'findModifiers,' to discover all available modifiers within the application, this includes specific bone heights, such as the 'femur,' and 'lower-forearm.' Once the initial subscript has been ran, a second script, 'createDataset,' is initialized. This script takes in the list of modifier variables, which are structured as such: "group_name/(action_being_modified)-(object-description)-(decr|incr)." The script organizes all modifiers into two categories: symmetrical, and non-symmetrical. Symmetrical modifiers include members that have an orientation designation, such as 'left,' or 'right,' for example: "armslegs/l-foot-scale-decr|incr." These modifiers are then sorted into pairs, through a hash-map.

Once everything has been sorted correctly, the script begins to generate random models, first the program calculates a randomized height, it is important to note that makeHuman doesn't characterize these modifiers by absolute measurements, rather by values between negative one and positive one, with zero representing the average. So, the program will need to utilize a decoder, to apply a representative value, for example 8'2", which is the tallest person alive, currently. Another important factor to realize, though every metric is relative, not absolute, calculations to find proportions will still be correct.

For example, to find the arm length, the program will take that randomly generated float value, representing height, some value (+/-)x.xx, and follow DiVicci's 'Vitruvian Man' ratio, which asserts that the arm span of an average man is equal to the total height of said man. With this formula, the

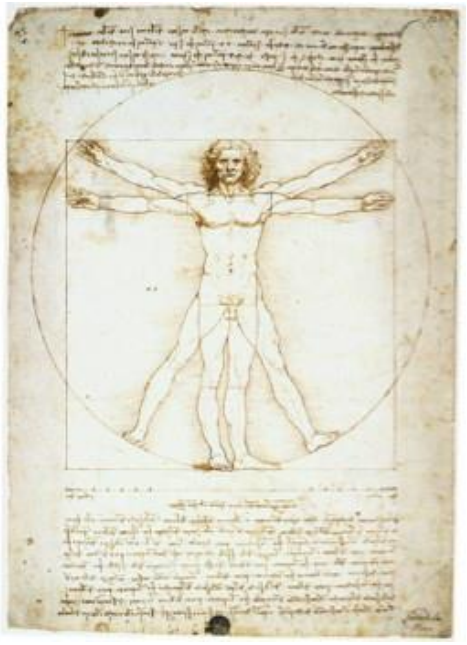


Figure 1.2.2:1 DiVicci's 'Vitruvian Man:' The ratios discussed correspond with DiVicci's diagram.

program can take the modifier: "macrodetails-height/Height," (total height) subtract: "measure/measure-shoulder-dist-decr|incr" (distance between shoulders), and divide the difference by two. The program, additionally adds some variation of noise to each factor, as the final result should not be an absolutely perfect ratio.

Continuing in this logic, the script, then finds the sub-ratio within each arm, referring back to the 'Vitruvian Man,' the ratio between the lower arm, the forearm, and the upper arm is about 1:1.2, or such that the lower arm, from elbow to finger tip is generally around forty-five and forty-eight percent of the arm. The script randomly generates a float value between these two values, then subtracting that from 1.00, the result being the length of the upper arm.

Using this logical methodology, relative lengths and sizing can be obtained and applied throughout the model.

Several other variables are kept, such as the muscle and fat variables, which keeps proportions of fat to muscle throughout the model standard, allowing slight variance, or standard deviation, which would be a randomized noise value, to again prevent these generated models from being perfect. With the general logic being, a person with high levels of muscle in their legs, will generally have large portions of muscle in their chest and arms.

Each individual model is saved unto the dataset as an .fbx object, though that is currently only for the proof of concept model, and may be altered as discussed later. Once this initial dataset is created, the Adobe software 'Mixamo,' is utilized to generate unique poses. A Python web scraper is ran, to access the *Mixamo* site, looping through the dataset and saving the animations into a new dataset, which is consequently brought into *Blender*.

Blender was chosen as this step's software because of it's scripting abilities that make distortion within a three-dimensional space straight forward. Blender's BPY, or the 'Blender Python API,' is well documented allowing updates and adjustments of the algorithm. Within Blender, a random value, between zero and one-hundred-twenty, the standard length of animations from Mixamo. The program then takes a key-frame of that random value, the current .fbx values of the model, the BPY script then selects a random HDRI, "High Dynamic Range Image," which acts as the lighting map for three-dimensional sub-spaces.

The script then renders the environment as a final output, a .jpeg extension, as well as the a final output, a .fbx extension. The images are mapped to their .fbx counterparts, this mapping takes the relative depth of the individual objects and translates them into absolute depth, so that model can learn to distinguish and predict physical depth in a real-world scenario.

Similar datasets for objects will implemented, and the overall and over-arching dataset will include: human, object, and structure classes.

1.2.3 Object:

To generate random objects, the model utilizes Blender, specifically BPY to run two sub-scripts as to interact with *SketchFab,* and *BlenderKit*, which are two large Blender plugin dependencies that house a combined total of over four million assets. Scripts in BPY, the Blender Python API, are highly interactive within Blender, that is, any action a user can do, can be done within the scripting tab. This includes choosing random assets to populate a scene.

Both SketchFab and BlenderKit organize their respective collections in Groupings. Such as: "Architecture", "Animal", "Structure → Interior". In this way, a environment is easily created, by building multiple templates, for example the dataset will need indoor settings, so we sort by architecture → Building → Industrial, where the last two keywords can be replaced by random values, to obtain a near infinite amount of random environmental settings.

Two scripts will be used, one for each Asset Library, to find the random objects we need, using the pre-made templates, think of this as procedural generation, or generating randomness using a template formula. These two scripts will be initialized by the main script, which is mentioned in the previous section, and is in charge of generating all the assets in our generated environmental dataset.

1.2.4 Structures:

Same as the scripts in the 'Object Creation,' two scripts, implementing BlenderKit and SketchFab are used to generate the structures, including buildings, floors and walls. However the key difference, is that the structures will be generated first, as this will determine both the placement of the humans and objects in the scene, as well as what objects should be included in the environment. For example, certain objects are, on average, less likely to appear indoors, such as a well, or fountain, wheelbarrow, and things of this nature. On the other hand, certain items can be reserved for indoors, such as refrigerators, mirrors, toilets, cabinets. Again the model's formulaic template will help handle these issues. The subscripts will sort the groupings with a list of common indoor items, and common outdoor items, the main script will then uses a random variable to decide if an environment should be indoor or outdoor, at the end the main script does a sanity check to ensure that both outdoor and indoor environments make up near fifty percent of the dataset.

1.2.5 HDRI's:

High Dynamic Range Images, or HDRI's, are image files that capture the lighting map of an environment. These are essential in rendering virtual environments that visually appear realistic. Again the model utilizes BlenderKit, which houses a large segment of HDRI's. In this manner, the script does not need to be highly specific with which HDRIs it surrounds the objects, humans, and structures with, the script only needs to randomly select from BlenderKit's assets. The sub-script does need to keep track of day maps, versus night maps, to ensure that the dataset has a near equal amount of both lighting. This is fairly easy to accomplish, as again BlenderKit is organized, specifically into groupings, which extends to it's HDRI assets. Once again, the main script will perform a sanity check, to ensure an adequate number of day and night environments have been selected and used.

2. Generative Adversarial Network:

2.1 Multi-class Classification:

The first step of the model, although the second addition and segment of the paper, is the multi-class classification, is the composite classification, which is to say separating the image into separate entities, to obtain the relative depth and measurements of objects. The reason for this composite logical reasoning is to increase applicable situations. If the model were to be trained by one layer, for example(an file of sheep in a field):

That situation is only useful for training once, however if the model is trained on specific objects separately, specifically in this case, sheep, the trees, and finally the field, it can be taught the relative sizing of common objects, and then apply that line of reasoning to spatial depth.

Therefore, a rudimentary image multi-class classifier can be utilized to determine what the model will need to deal with. In this convolution approach, take again our example of sheep, can classify the sheep, remove those pixels from the image, and continue onwards until the image in its entirety has been classified.

Once an object has been classified by the model, and the model has identified the corresponding label and class, the corresponding GAN model and dataset may be selected and implemented. Using three separate models should promote an increased accuracy and efficiency at the cost of longer implementation time during the initial set up and building of the model.

That is to say, when the model is first being built a longer duration will be needed to

train the model, but this should provide more accurate predictions in depth and measurements, as even if the model's prediction isn't one hundred percent accurate, there are set outlying values, for example, in the human class, as previously discussed, the largest height and width could not be larger than 8'2", which should lower hallucinations, and provide some form of safe guard against outlier predictions, which could not be achieved by using a single solid dataset.

Pytorch's 'ResNet50' is the proposed model, due to its simplicity, and commonality. Though no Convolution Neural Network should be classified as simple, ResNet has the distinct advantage of being thoroughly well documented by *Meta's* AI department, making altercations and updates straightforward if, and when, changes and revisions must be made.

2.2 Absolute Depth Estimation using GAN's:

Three separate and independent generative adversarial networks are to be implemented and trained, a separate network for each corresponding class set, object, human, and structure. The multi-class classifier will pass the output of each sweep pass of the input file to the complementary network, and here is where the greatest reactionary effect of using a virtually simulated dataset occurs, since the dataset only deals with the relative measurements of objects, the GANs predict those relative depths, for example, take the sheep example, once more, the model can determine that input file's sheep is five foot six inches from head to tail, and can then apply this line of reasoning to the next class. For example, determining how far the camera is from the next object, essentially turning common objects into measuring utensils.

These GANs are not necessarily equivalent in terms of hierarchy. That is objects and humans must always be determined in sequential order, preceding the classification of structures. As seen in the figure below, the structure is organized and implemented in such a way that the model's encoder resides before the initialization of the structural GAN, such that the image may be put back together by the encoder before the internal three-dimensional data is given to the final structure GAN.

2.3 Data Encoder:

This is such that the aft-fore mentioned process can occur, the encoder will either use the "pixel-per-metric" method, such as learning that an apple is five inches, so if an apple is resting on top of a table, that it is reasonable to assume that both reside on the same axis, and a relatively accurate measurement prediction can be made of the table.

Again, the use of composite sweep analyses of the image input allows a more accurate resulting output prediction of individual objects.

In terms of foreign concepts, that is objects that were not captured into the dataset, and therefore the model could not reliably predict the foreign objects' surface area, a general measurement can be obtained, by using the encoder to compare the sizes of known objects to unknown objects. The encoder is either implemented by the *OpenCV* Python Library, or as mentioned prior, "pixel-per-metric."

The encoder is the integral machination of the model and meshes the classification with the generative prediction between the multiple models, allowing the model to understand its surrounding environment in three-dimensional terms.

After combining the many separate

Figure 2.4.1:1: Example of what the mapping process might produce.

predicted output layers, the encoder pieces the image back together.

2.4 Siamese Neural Network:

Taking the prediction probability of the encoder, which is the average output probability of each of the separate GANs, the output of the encoder is saved into a file, where a BPY sub-script will populate a scene in blender, an empty three-dimensional environment, with the output of the three GAN's as well as looping through the HDRI dataset.

A threshold is given to the model, for example, ninety-three percent, anything under this value will be run through the SNN model, here the rendered image output of the encoder is inputted through the SNN, as well as the initial input image.

Each pixel in both input images of the SNN, are vectorized into R^3, and using Euclidean distance to find comparison between the two inputs, resulting in a probability score. The proposed SNN model is the ResNet SNN, for consistency. The reasons to use ResNet were listed prior, ResNet also provides a SNN.

For this model, the encoder head of the ResNet SNN model will be disconnected and removed, leaving the classifier architecture. It is important to note, that the SNN, ideally, should not be used for every prediction outcome, only in outlying cases, when the model is unable to accurately depict the input in a three-dimensional setting.

2.4.2 Manual Backpropagation:

The Siamese Neural Network will be used as an advanced error detection, the output of the SNN is an embedding, informing the model which attributes of the input are important, and from there the model can determine what went 'wrong,' with its initial prediction.

Using the embedding the model may find the axis that is the most ill-matched between the two inputs, from there the model can determine the corresponding attribute, such as the object or structure from the generated rendered GAN image output, and traverse backwards, and make a new prediction on the initial multi-class classifier input.

2.5 Projection Mapping(optional):

Once an accurate generative prediction has been formed from the initial input image,

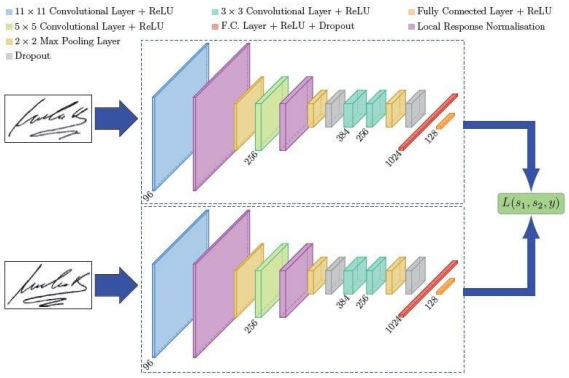


Figure 2.4.2: The architecture of a standard Siamese Neural Network, where the top sub-structure represents the initial input image, and the bottom represents the output of the encoder.

that is a generative prediction being a

singular file output, .fbx extension, the combined meshes, or objects, from the all the independent GANs, a final layer function will 'wrap' the completed object file. Similar to 'UV mapping,' the program takes the initial input image and projects it unto the final three-dimensional scene.

In UV mapping, a flat two dimensional image, with axes: "U(horizontal)," and "V(vertical)," holds the image texture that will wrap around an image. The image contains 'seams,' which represent the edges of the object, essentially a visual formula to map two dimensional unto three dimensional, where the image, represented by some vector in R^2 is transformed into a vector in R^3.

This process is close to what is require here, but not exact. As can be seen below, in figure **2.5.1**, the generated three dimensional space will not be completely precise, this method ideally requires a one-hundred percent accuracy, if you recall, ninety-three percent is what this algorithm aims for. Leading to stray artifacts in the texturing. Alternatively, in the proposed method, projection mapping, the algorithm assumes that the virtual camera is positioned at the root origin, <0,0,0>, and can iterate through the initial image input, creating a ray per pixel, which is to say, take the camera position, <0,0,0>, C_1, the intended target, $<x_1,y_1,z_1>$, T_1, Subtract T_1 by C_1, to find the direction vector, or magnitude vector, d_1. Then the program initializes a step variable, how much the direction vector is incremented by, *t.* This gives the final ray: ray = $C_1 + tT_1$. Using the ray, we can then 'project,' the initial image texture back unto the three-dimensional object.

This method is optional, and is used as a polishing function, to ensure that the final output most closely aligns with the inputted image, the goal being that predicted depth, and the corresponding textures of the data set help blend the projected texture with the generated texture.

2.5.2 Generated Textures:

The former process does have some clear obstacles, though the original image texture is projected unto the object file, that covers only one viewpoint. The program needs to assign and generate textures for the geometry that is not visible in the initial image. As can be recalled from section **2.1**, all the object assets in an image are already being sorted by the model, the program will keep a separate dataset, whose only function serves to generate textures, each holding matching image textures for each of it's corresponding datasets: structures, humans, and objects. In this way, the algorithm can make assumptions on what an object is, and fill in unseen portions of the viewport with these well of image textures.

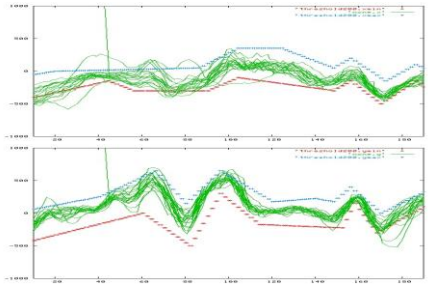


Figure 2.5.2:1: The output of both gyroscope measurements, during the walking gait cycle, (top graph: sagittal plane measurements, bottom graph: horizontal plan measurements)

It is important to note all of segment **2.5**, is optional as is not needed for every application of the algorithm, see: **2.6: Real World Applications**. For example, using this algorithm for real world interactions, it is not necessary to generate textures, as the generated depth prediction is to be used as a map, for the model to correlate an object with some depth estimation in R^3. For this

reason, a 'Generative Textures' dataset was not included in section 2.

3. Real World Applications:

3.1 Humanoid Robotics, operating systems in R^3:

Implementing CARLA into a humanoid robot can be achieved using a combination of previously researched balance methods. Using the two gyroscopic setup discussed in the '**Active Balancing Using Gyroscopes for a Small Humanoid Robot**,' which uses two gyroscopes to determine "safe zones," or which is the range of velocity that 'TAO-PIE-PIE,' the robot proposed in that particular research thesis, utilizes for its walking gait. For integration of the CARLA algorithm, the proposed method is to replace the 'Sagittal Gyroscope,' the gyroscope atop the robot's 'head,' with the model algorithm.

The lateral gyroscopic reading metric is kept, the gyroscope is placed around the hips, to measure lateral plane measurements. In this way, the robot can understand its surrounding environment, able to differentiate between separate composite objects, while also keeping a gyroscope for its lower stability.

Additionally, using a '**Design of an Anthropomorphic Robotic Finger System with Biomimetic Artificial Joints**,' the concept of a humanoid robot being able to freely interact within its environment becomes substantially more plausible. In this methodology, the traditional joints are replaced by elastic based bio-joints, allowing more range of motion in all three dimensions, while maintaining grip strength. The proposed concept is to use the CARLA algorithm as the interface for the biomimetic hand, allowing the robot to understand its surroundings, and interact in real time, on its own design.

Combining these ideas, a robotics initiative can be imagined where the creation is able to move and learn about its setting, using this approach, the idea of generating textures can be dismissed, as mentioned in **2.5.2**, as the algorithm needs only to understand the predicted depth values of individual objects in its environment, that is, understanding that some subsection of pixel values represent an independent object, that may be interacted with.

Subsequently, a modified version of the algorithm can be approached, where instead of image input, a LiDAR, light detection and ranging system can be substituted. Here, the LiDAR would be used to get the initial three-dimensional measurements, while the algorithm would provide what the visible objects non-visible portions would be. For example, if the system was given input from the LiDAR, which uses a Point cloud file type, the algorithm could then begin the same process of breaking down the input, this time the input is already in a three-dimensional space. This would highly improve efficiency and accuracy for the model. However, a substantial amount of revisions would need to be made to the algorithm, particularly to the datasets.

New datasets would need to be created, as, although the previously proposed datasets are created in a three-dimensional space, they are not formatted in a point cloud file type. Point clouds represent data points by a coordinate system, whereas in a fbx file type, three-dimensional objects are represented by vertices and edges. This introduces several new challenges, as mentioned prior, initial accuracy in determining the depth of objects is increased substantially, however classification of independent objects is highly diminished. This is easy to understand when one thinks about how the objects can be separated, often along edges, or seams, between vertices. However, in

point clouds there are no edges, so additional work must be done to distinguish between composite layers.

This tradeoff is completely dependent on what function is needed, whether precision, or interaction. This must be decided on a per-project basis, depending on the needs of the creation intended. The work to revise the algorithm would include updating the datasets, training a new GAN system, essentially reforming the entire algorithm to work with point cloud data. This is most certainly a large undertaking, but not unfeasible. It is important to note that all previous software's in this paper are capable of creating point cloud assets and data. Once again, the main issue will be distinguishing between objects from an initial input file.

3.2 Production in Media and Visual Effects:

Incorporating the ideas introduced in segment **2.5.2**, a robust system for transferring on set props and performances into a virtual three-dimensional setting. In this use case, the algorithm is used on set during a performance, where actors performance can be captured directly, instead of post-processing, the algorithm can be used to recreate scenes, where visual effects artists can acquire precise depictions of the production's real world set and characters.

Here, generating textures would be mandatory, as being able to modify the cameras in a virtual space would be essential in improving efficiency and cost saving procedures for production in film, this being the essential draw of using the algorithm in this manner. The algorithm must still be modified, as the best results would come from the model being able to understand video, rather than specific key frame moments. To do this significant memory must be introduced to the system, specifically temporal memory, that is to say frame order. This will help train the model, and help it understand movement, and give it the ability to refer backwards to earlier frames.

This is paramount to understanding motion blur, when objects, or people enter or exit the visible frame. Motion blur is large issue, and the model must understand the concept of not losing or gaining mass during aggressive movements. For example, a character running, which surely will cause motion blur, the model needs to understand that the legs of a character are not gaining mass as motion increases, so the model needs to call back to earlier definitions of measurements and initial predictions.

The benefit here is that no specific modifications to camera equipment or software need to be made. Once the footage is filmed, it can be transformed into a three-dimensional file, and opened and modified in any three-dimensional virtual environment, for example, *Autodesk Maya, Blender, Houdini,* and *Unreal Engine*.

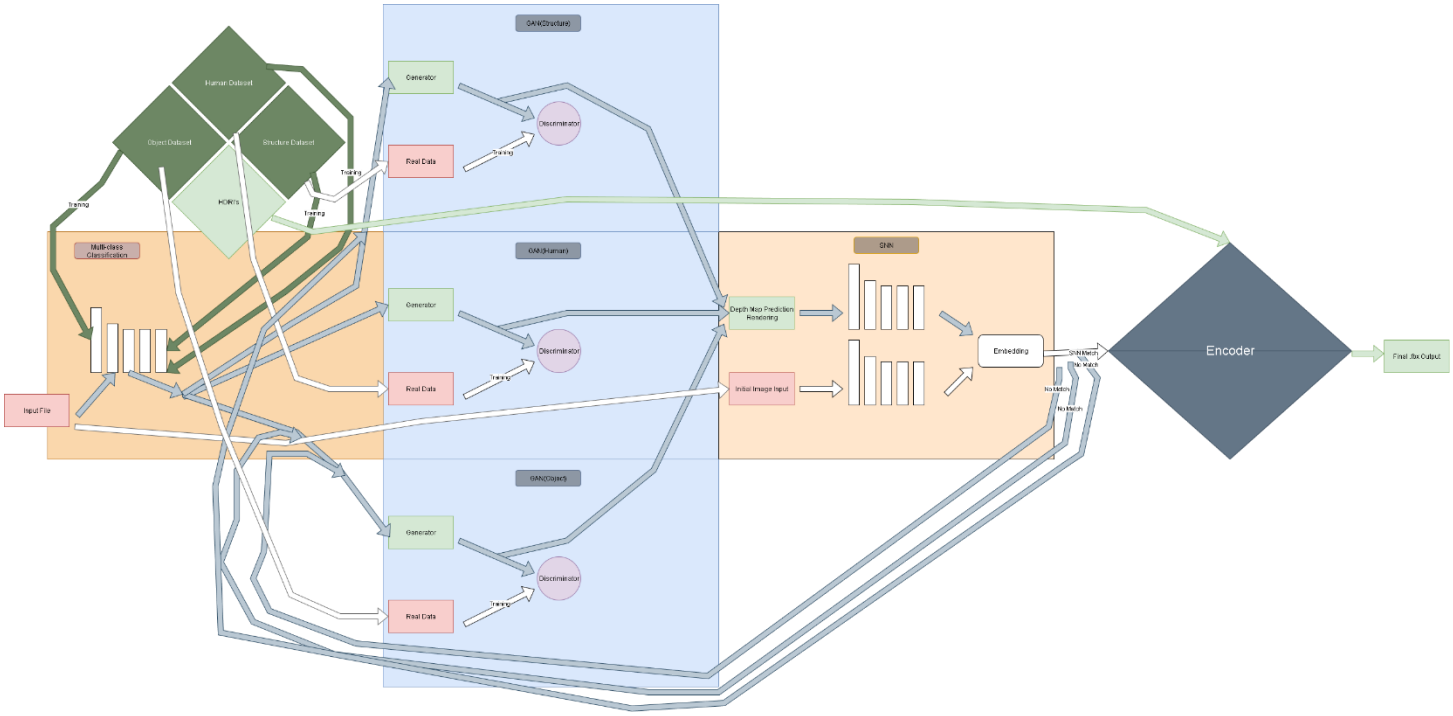
4. UI and Application:

A software application will be built to house the algorithm, this will be a three-dimensional environment that can display the output of the model, this can be achieved using a combination of the following: *Matplotlib, Open3D, and Panda3D*. The application will be used to run the algorithm on selected input files, as well as display the predicted output. Minimal editing and modification operations will be included. In this way, the software is mostly meant to perform a single determining function, and not to replace any three-dimensional editing software's.

Subsequently, several plug-ins and add-ons may be created, that work within other software's, such as: *Autodesk Maya,*

Blender, Houdini, and *Unreal Engine*. These plug-ins, depending on the software, are fairly easy to create and script. The main factor in this decision, is that users do not need to learn new software, but can rather continue using software they are more familiar with, in this way, the algorithm is a tool, rather than a creative environment.

4.1 Algorithm Flow of Data:



References:

Danie Conradie, et al. “Actively Balancing a Robot with a Gyroscope.” *Hackaday*, 16 May 2021, hackaday.com/2021/05/17/actively-balancing-a-robot-with-a-gyroscope/.

Dutta, S., et al. “Stability Analysis of Humanoid Robots with Gyro Sensors Subjected to External Push Forces | IEEE Conference Publication | IEEE Xplore.” *Stability Analysis of Humanoid Robots with Gyro Sensors Subjected to External Push Forces*, IEEE, 2019, ieeexplore.ieee.org/document/8719090.

Baltes, J., McGrath, S., & Anderson, J. (2004). Active Balancing Using Gyroscopes for a Small Humanoid Robot. *2nd International Conference on Autonomous Robots and Agents*.

Xu, Zhe, et al. “Design of an Anthropomorphic Robotic Finger.” *Roboti.Us*, www.roboti.us/lab/papers/XuBioRob12.pdf. Accessed 7 Jan. 2026.

“Enjoy 54,631 Free Models, Materials, Hdris & More.” *BlenderKit*, www.blenderkit.com/about-blenderkit. Accessed 7 Jan. 2026.

Ida, Takashi. “‘Vitruvian Man’ by Leonardo Da Vinci and the Golden Ratio.” *Web Site of Takashi IDA*, Advanced Ceramics Research Center, Nagoya Institute of Technology, Japan, 2012, takashiida.net/index/en/education/vitruvian-man/.

Sketchfab, sketchfab.com/feed. Accessed 7 Jan. 2026.

Kruk, Mariusz. *The Eurocall Review, Volume 25, No. 2, September 2017 18 Research Paper*, University of Zielona Gora, Poland, 2017, files.eric.ed.gov/fulltext/EJ1172284.pdf.

Hendrycks, Dan, et al. “Superintelligence Strategy: Expert Version.” *arXiv.Org*, 14 Apr. 2025, arxiv.org/abs/2503.05628.

Mokhtari, Nour Islam. “What Are Siamese Neural Networks in Deep Learning?” *Towards Data Science*, 16 Jan. 2025, towardsdatascience.com/what-are-siamese-neural-networks-in-deep-learning-bb092f749dcb/.

“Resnet50¶.” *Resnet50 - Torchvision Main Documentation*, docs.pytorch.org/vision/main/models/generated/torchvision.models.resnet50.html. Accessed 7 Jan. 2026.

Rosebrock, Adrian. “Measuring Size of Objects in an Image with Opencv.” *PyImageSearch*, 20 Dec. 2024, pyimagesearch.com/2016/03/28/measuring-size-of-objects-in-an-image-with-opencv/.

Original Proof of Concept(not updated):

Overview: