

DEVELOPMENT OF SYNTHETIC AND REAL-WORLD POSE ESTIMATION
DATASET TO BE USED IN HUMAN TRACKING SYSTEM

A THESIS SUBMITTED TO
THE GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES
OF
MIDDLE EAST TECHNICAL UNIVERSITY

BY
MUSTAFA ERSOY

IN PARTIAL FULLFILLMENT OF THE REQUIREMENTS
FOR
THE DEGREE OF MASTER OF SCIENCE
IN
MECHANICAL ENGINEERING

APRIL 2022

Approval of the thesis:

**DEVELOPMENT OF SYNTHETIC AND REAL-WORLD POSE ESTIMATION
DATASET TO BE USED IN HUMAN TRACKING SYSTEM**

Submitted by **MUSTAFA ERSOY** in partial fulfillment of the requirements for the degree
of **Master of Science in Mechanical Engineering Department, Middle East Technical
University** by,

Prof. Dr. Halil Kalıpçılar
Dean, Graduate School of **Natural and Applied Sciences**

Prof. Dr. M. A. Sahir Arıkan
Head of the Department, **Mechanical Engineering**

Associate Prof. Dr. Ahmet Buğra Koku
Supervisor, **Mechanical Engineering, METU**

Examining Committee Members:

Prof. Dr. Yiğit Yazıcıoğlu
Mechanical Engineering, METU

Associate Prof. Dr. Ahmet Buğra Koku
Mechanical Engineering, METU

Assistant Prof. Dr. Ali Emre Turgut
Mechanical Engineering, METU

Assistant Prof. Dr. Emre Akbaş
Computer Engineering, METU

Assistant Prof. Dr. Kutluk Bilge Arıkan
Mechanical Engineering, TED University

Date: 29.04.2022



I hereby declare that all information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all material and results that are not original to this work.

Name, Surname: Mustafa Ersoy

Signature:

ABSTRACT

DEVELOPMENT OF SYNTHETIC AND REAL-WORLD POSE ESTIMATION DATASET TO BE USED IN HUMAN TRACKING SYSTEM

Ersoy, Mustafa

Master of Science, Mechanical Engineering

Supervisor: Associate Prof. Dr. Ahmet Buğra Koku

April 2022, 93 pages

In this study, we propose an extendable, synthetic human pose estimation dataset named “Metupose”. Pose estimation aims to determine the pose of a person by detecting joints in an image or video. Dataset was created in Blender 3D software and with varying human objects and environment. It is also used to enhance the accuracy of pose estimation models in the literature. Metupose dataset contains 178000 images. Images have 1 to 4 people in it, where there is a total of 402000 people exist in these images. When we train different pose estimation models from the literature with our dataset, we observed an accuracy increase in all model/dataset cases. Our second contribution is the source code to create new images for the dataset. Although, Metupose contains large number of images for most of the applications, users may need to create their own custom dataset or want to increase the number of images. We provide original Blender 3D files and a simple configuration file so that users can create new dataset easily. Normally, creating a real-world dataset is time consuming and open to labelling errors. The advantage of our study is that datasets of desired sizes can be created from software, without any error, in an automated way. We finally, trained a pose estimation model with our Metupose dataset and integrated the trained model into Nvidia Jetson Nano that is equipped with a Raspberry Pi Camera and evaluated human tracking performance. Results indicate that single board computers offer a low-cost alternative to be used in human-robot interaction studies.

Keywords: Pose Estimation, Extendable Dataset, Synthetic Dataset, Human Tracking

ÖZ

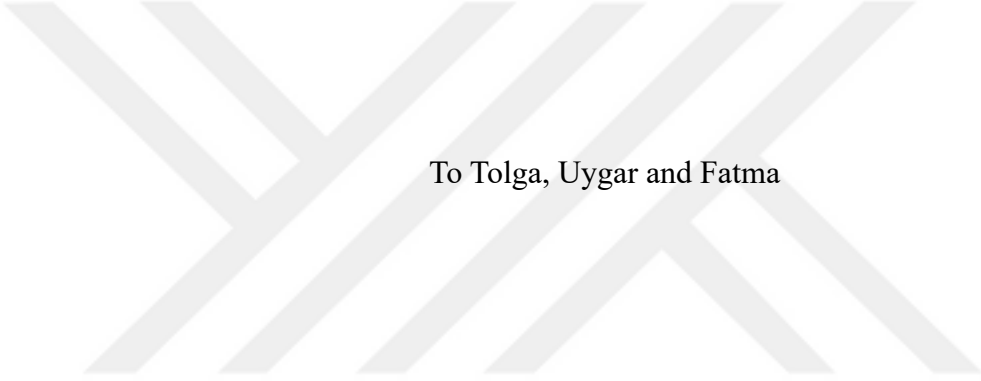
DEVELOPMENT OF SYNTHETIC AND REAL-WORLD POSE ESTIMATION DATASET TO BE USED IN HUMAN TRACKING SYSTEM

Ersoy, Mustafa
Yüksek Lisans, Makina Mühendisliği
Tez Yöneticisi: Doçent Dr. Öğretim Üyesi Ahmet Buğra Koku

Nisan 2022, 93 sayfa

Bu çalışmada, “Metupose” isimli sentetik ve genişletilebilir insan iskelet takibi veri seti sunduk. İskelet takibi, bir görüntü veya videodaki insan eklemlerini algılayarak insan duruşunu algılamaktır. Veri seti, Blender 3D programında değişken insan modelleri ve çevreye sahip şekilde üretilmiştir. Ayrıca, literatürdeki iskelet takibi modellerinin doğruluğunu artırmada kullanılmıştır. Metupose veri setinde 178000 görüntü vardır. Her bir görüntüde 1,2,3 veya 4 insan olmak üzere toplamda 402000 insan bulunmaktadır. Farklı iskelet takibi modellerini veri setimizle eğittiğimizde tüm model/veri seti durumlarında doğruluk artışı gözlemledik. İkinci katkımız ise, veri setinde yeni görüntüler oluşturmaya yarayan kaynak kodudur. Metupose veri setinde birçok uygulama için yeterli görüntü olsa da kullanıcılar kendilerine ait veri seti oluşturmak isteyebilir veya görüntü sayısını artırmak isteyebilir. Kullanıcıların kolayca yeni veri seti oluşturabilmesi için, orijinal Blender dosyalarını ve basit konfigürasyon dosyasını yayımladık. Normal koşullarda, gerçek hayat veri seti üretmek çok zaman alabilir ve hatalara açıktır. Çalışmamızın avantajı, sadece yazılım üzerinden otomatik şekilde hatasız veri seti üretmeye imkân sağlamasıdır. Son olarak, Metupose veri seti ile eğittiğimiz iskelet takibi modellerini Raspberry Pi kamerasına sahip Jetson Nano’da çalıştırdık. Sonuçlar, bu sistemin, insan-robot etkileşimi çalışmalarında düşük maliyetli bir alternatif olabileceğini gösterdi.

Anahtar Kelimeler: İskelet takibi, Genişletilebilir Veri Seti, Yapay Veri Seti, İnsan Takibi



To Tolga, Uygur and Fatma

ACKNOWLEDGEMENTS

The author wishes to express his deepest gratitude to his supervisor Associate Prof. Dr. Ahmet Buğra Koku for his guidance, advice, criticism, encouragements, and insight throughout the research.

I would like to extend my thanks to ODTÜ BAP Koordinasyon Birimi for their financial support in this study. (Project No: 10587)

I would like to thank METU ROMER for letting me perform experiments on their Vicon system during my studies.

I would like to extend my thanks to Fatih Tosun for his friendship and for supporting me every day during my studies.

I sincerely thank to my friend Burak Anıl Güler for his friendship and everything. Also, for convincing me to buy his GPU during my studies. Without that purchase, this study would not be possible.

I also would like to thank to my old team manager Dr. Ahmet Bayrak and my coworkers Beyza Nur Cenkci, Mete Gökalp and Sarp Karadayı for always supporting and believing in me.

And of course, my greatest thanks go to Ersoy family including my parents Murat Ersoy, Ayfer Ersoy and my brother Tolga Ersoy, my uncle Duran Ersoy and his wife Ümmühani Ersoy and my cousins Uygur Ersoy and Fatma Ersoy. You all always supported and believed in me and encouraged me to push my limits in my entire life. Thank you all for everything.

TABLE OF CONTENTS

ABSTRACT	v
ÖZ	vi
ACKNOWLEDGEMENTS.....	viii
TABLE OF CONTENTS	ix
LIST OF TABLES.....	xii
LIST OF FIGURES	xiii
CHAPTERS	
1. INTRODUCTION.....	1
1.1. Problems in Pose Estimation	1
1.1.1. Occlusion.....	2
1.1.2. Unknown Number of People	2
1.1.3. Interaction Between Different People	3
1.1.4. Decrease in Running Speed with Increase in Number of People	3
1.1.5. Accurate External Camera Calibration in Stereo Images	3
1.1.6. Not Having Enough Data and Errors in Labelling of Data	4
1.2. Literature Survey	4
1.3. Human Pose Estimation Datasets	6
1.3.1. COCO Dataset.....	6
1.3.2. MPII Dataset.....	7
1.3.3. H36M Dataset	7
1.3.4. CrowdPose Dataset	8
1.3.5. LSP/MPII-MPHB Dataset.....	8
1.4. Problems with Pose Estimation Datasets	9
1.5. Synthetic Datasets.....	10
1.6. Synthetic Datasets in the Literature	12
1.6.1. SurReal Dataset.....	12
1.6.2. JTA Dataset.....	13
1.6.3. AGORA Dataset	15
1.7. Motivation	16

1.8.	Layout of Thesis.....	17
2.	PERFORMANCE of POSE ESTIMATION MODELS on SYNTHETIC IMAGES.....	19
2.1.	Vrep 3D Software Information.....	19
2.2.	Performing Pose Estimation on Different Image Groups.....	21
2.3.	Quantifying Performances of Pose Estimation Models.....	24
2.3.1.	Perfect Label.....	24
2.3.2.	Fair Label.....	24
2.3.3.	Bad Label.....	25
2.3.4.	No Detection Label.....	25
2.3.5.	Wrong Detection (Extra person detection).....	25
2.4.	Domain Randomization.....	30
2.5.	Final Notes.....	31
3.	STUDIES ON BLENDER 3D SOFTWARE.....	33
3.1.	Blender General Information.....	33
3.2.	Blender Properties and Capabilities Regarding Pose Estimation.....	34
3.2.1.	Creating Human Models.....	34
3.2.2.	Person Rigging and Animation.....	37
3.2.3.	Python API.....	38
3.2.4.	Changing Appearance of Human Model.....	38
3.3.	Image Creation in Blender.....	41
3.3.1.	Creating a Street Environment.....	41
3.3.2.	Human Model in Front of a Plane Video Method.....	43
3.3.3.	Person Cropping Method.....	46
3.3.4.	Pose Complexity Analysis.....	55
3.3.5.	Further Details About Three Dataset Generation Methods.....	64
3.4.	Testing Different Pose Estimation Models on Blender Dataset.....	66
3.5.	Creation of Blender Dataset.....	68
4.	TRAINING MODELS with BLENDER DATASET.....	71
4.1.	Blender Annotation File Details.....	71
4.2.	Training Models on Blender Dataset.....	72

4.3. Details about Training.....	73
4.4. Deploying the Trained Model on Jetson Nano	81
CHAPTER 5	85
5. CONCLUSION.....	85
5.1. General Conclusion	85
5.2. Future Work.....	86
REFERENCES.....	87
6. APPENDIX	93



LIST OF TABLES

TABLES

Table 1: Name of Joints Used in Datasets in Literature	5
Table 2: Sample Images from Different 3D Design Software	11
Table 3: Comparison of Surreal and Metupose Dataset	13
Table 4: Comparison of JTA and Metupose Dataset	14
Table 5: Comparison of AGORA and Metupose Dataset.....	15
Table 6: Sample Images for Labels Used to Evaluate Outputs	26
Table 7: Comparison of Three Different Pose Estimation Models on Real Data Group by Percentage	27
Table 8: Comparison of Three Different Pose Estimation Models on GTA5 Data Group by Percentage	27
Table 9: Comparison of Three Different Pose Estimation Models on Unity Data Group by Percentage	28
Table 10: Comparison of Three Different Pose Estimation Models on Vrep Data Group by Percentage	28
Table 11: Accuracy of Model 1 Throughout Different Data Groups by Percentage	29
Table 12: Sample Images for Textures in Blender	39
Table 13: Content of Our Source Code	52
Table 14: Number of Vectors in Each Category, Group.....	60
Table 15: Performance of 4 Different Models on Blender Dataset by mAP metric.....	67
Table 16: Order and Name of Joints in Blender .csv Dataset.....	71
Table 17: 3 Experiments to measure effectiveness of Metupose dataset	74
Table 18: Accuracies of Pretrained Github Models on 4 Different Datasets	76
Table 19: Explanation for Different Datasets	77
Table 20: Explanation for Different Training Conditions	78
Table 21: Definition of Sample Pose Estimation Model Names	78
Table 22: Accuracy Values for Experiment 1 (Pose ResNet50 + MPII Dataset)	78
Table 23: Accuracy Values for Experiment 2 (Pose ResNet50 + Crowdpose Dataset) ...	79
Table 24: Accuracy Values for Experiment 3 (HRNet + MPII Dataset)	79
Table 25: Performance of Metupose Dataset in Different Configurations.....	80
Table 26: Real Time Fps Values of Our Trained Pose ResNet50 Model in Different Hardware	82
Table 27: Real Time Fps Values of Our Trained Pose HRNet Model in Different Hardware	82

LIST OF FIGURES

FIGURES

Figure 1: Sample Annotated Image from COCO Dataset [7]	6
Figure 2: Sample Images from MPII Dataset [4].....	7
Figure 3: Sample Images from H36M Dataset [6].....	8
Figure 4: Sample Images from Crowdpose Dataset [8].....	8
Figure 5: Sample Images from LSP/MPII MPHB Dataset [9]	9
Figure 6: Sample Images from Surreal Dataset	13
Figure 7: Sample Images from JTA Dataset	14
Figure 8: Sample Images from AGORA Dataset	16
Figure 9: Human Models in Vrep Software	20
Figure 10: Outputs of Github Pose Estimation Model on images from Vrep software ...	21
Figure 11: Pose Estimation Model Outputs on Unity Human Model Images	22
Figure 12: Pose Estimation Model Outputs on GTA5 Images.....	23
Figure 13: Pose Estimation Outputs for Real Human Images from TV Shows.....	23
Figure 14: Blender Human Models in Turbsquid Website [26]	35
Figure 15: Default Human Model Created by MbLab addon in Blender	35
Figure 16: Demonstration of 6 Steps to Create a Human Model from Scratch	36
Figure 17: Three Different Poses of the Same Human Model and Joints.....	37
Figure 18: Human models created by different textures	40
Figure 19: Sample Image for Street Model Created in Blender.....	42
Figure 20: Human Models Placed in the Street Model in Blender	43
Figure 21: Outside view of Street Model in Blender	44
Figure 22: Human Model in front of a Video Plane (Rendered View)	45
Figure 23: Human Model in front of a Video Plane (Outside View)	45
Figure 24: Sample Images with Changing Background	46
Figure 25: Human Model, Cameras and Lights inside 100 Meters Diameter Green Sphere in Blender	47
Figure 26: Left Side: Images Rendered from 3D Blender Space, Right Side: Cropped Version of Images in Opencv	48
Figure 27: Cropped Human Model Images Pasted onto Variable Background	49
Figure 28: Various Objects Added to Image to Create Occlusion	50
Figure 29: Rendering Images from Blender	54
Figure 30: Cropping Human Models from Green Screen Images	54
Figure 31: Sample Image Showing a Joint Vector on a Human Model	57
Figure 32: Joint Vectors for Left Shoulder to Elbow in Entire Dataset with 10 Images..	58
Figure 33: Grouping of Joint Vectors Based on 20x20 Grid Categories.....	59
Figure 34: Histogram of Scores for 4 Person Case Using 30000 Images	61
Figure 35: Histogram of Scores of Cropped Person Library (49000 Images)	61
Figure 36: 5 Images with Highest Pose Complexity Scores	62
Figure 37: 5 Images with Lowest Pose Complexity Scores.....	62

Figure 38: Two Different Keyframes from Blender Animation	65
Figure 39: 3D to 2D Camera Projection Procedure [31]	68
Figure 40: Representation of Pinhole Camera Models (HeadVision, 2019)	70
Figure 41: Architecture of Pose ResNet50 models used in Training [36]	74
Figure 42: Framework of the Pose ResNet Model Used in Training [36]	75
Figure 43: Architecture of pretrained HRNet Model	77
Figure 44: Throttle Error on Jetson Nano During Real Time Pose Estimation via ResNet50 Model.....	83



LIST OF ABBREVIATIONS

ABBREVIATIONS

PCK: Percentage of Correct Keypoints

PCKh: Percentage of Correct Keypoints-head

mAP: Mean Average Precision

2D: Two Dimensional

3D: Three Dimensional

OS: Operating System

CPU: Central Processing Units

GPU: Graphics Processing Units

f_x : Focal length of camera in x direction

f_y : Focal length of camera in y direction

u, v : pixel coordinates

c_x, c_y : optical center location of camera

X_c, Y_c, Z_c : 3D location of an object in camera coordinate system

X_p, Y_p, Z_p : 3D location of an object in world coordinate system

s : scaling factor in camera



CHAPTER 1

1. INTRODUCTION

Human tracking is one of the most important concepts in computer vision. It is used for both human-computer communication and acquiring data from human movement. One of the ways to perform human tracking is the method called pose estimation. In this method, physical joints of people in each image are found and labelled by the algorithm. Depending on the pose estimation algorithm or model, selected joints may vary. However, most common ones are head, neck, shoulders, elbows, wrists, hips, knees, ankles. Because pose estimation is a difficult computer vision problem to model mathematically, its algorithm cannot be coded from scratch. Instead, deep learning models are trained on so many images with labeled data. When the deep learning model is fully trained, it is tested on new, real-world data to evaluate performance. In this study, we propose a pose estimation dataset named as “Metupose” and the source code to create new, custom-made dataset. Metupose dataset was used to train different pose estimation models and accuracy of trained models increased. Finally, trained models were integrated into Jetson Nano and Raspberry Pi camera system to be used in human tracking applications.

In this chapter, general problems in pose estimation models and dataset have been discussed and literature survey has been provided. Secondly, our solutions to these problems and our motivation have been explained.

1.1. Problems in Pose Estimation

Although there is so much research conducted on pose estimation, there are still minor problems in this field. These problems can be listed as follows:

1. Occlusion
2. Unknown number of people in an image

3. Interaction between different people
4. Decrease in running speed with increase in number of people
5. Accurate external camera calibration in stereo images
6. Not having enough data and errors in annotation of data

1.1.1. Occlusion

This problem occurs when test object stays behind of an object and some of the joints are not visible due to occlusion by another object. The reason behind this problem is that most of the datasets consist of full human body images. When models are trained on images with full human body, they may not detect occluded parts very well. To avoid these problems more occluded images can be used for training.

1.1.2. Unknown Number of People

Some pose estimation models are used for single person pose estimation and most of them are used for multi-person pose estimation. Single person pose estimation models such as EfficientPose [1] try to find only 1 person in an image fit keypoints into found person object. Keypoints can be considered as joints (shoulder, knee, neck etc.) in human body. Multi-person pose estimation models such as BaPose [2] however, try to find number of people correctly and try to fit keypoints into people found in the image. Some pose estimation models use highly accurate person detection or person segmentation models to locate each person effectively by fitting in a bounding box, then, perform pose estimation in that box to increase accuracy. However, when number of people increases in an image, the model may guess number of people in an image wrong and assign keypoints to wrong person. The reason behind this problem is that datasets may not cover different number of people in an image and also may not cover all possible interactions between multiple people.

1.1.3. Interaction Between Different People

In multi-person images, some models may assign one person's joints to another person. The reason can be two people can be so close to each other or there may be an interaction between two person such as occlusion. The reason behind this problem is that datasets may not cover all possible and complex interactions between people in an image. Therefore, when a model is tested on image of two very close people, model may assign one person's arm to another.

1.1.4. Decrease in Running Speed with Increase in Number of People

This problem occurs mostly in models trained with top-down approaches. Top-down approaches such as RMPE [3] perform human detection or human segmentation firstly and create a bounding box for each person, later perform pose estimation in each box. In this approach, computation increases linearly with increasing number of people because the model detects each person one by one perform pose estimation on each person one after another. In top-down approaches, although the running speed decreases, pose estimation accuracy increases.

1.1.5. Accurate External Camera Calibration in Stereo Images

If properly implemented, 3-D pose estimation gives more valuable data than 2-D pose estimation. To train 3-D pose estimation model, one needs to have 3-D data of keypoints. Proper way of doing that is using multiple images. When multiple images are used and both internal and external properties of each camera is available, 3D location of a joint can be found by 3D reconstruction method. However, external parameters of multi camera setup needs to be known and it is difficult to get position and orientation of each camera outside laboratory. Most of the multi-camera datasets are created in laboratory but not outdoor. In laboratory the background is always same and because of that, models may

not work properly when tested with real world data with completely different background.

1.1.6. Not Having Enough Data and Errors in Labelling of Data

A pose estimation model should be able to work in so many domains with high accuracies to be considered as effective. One of the ways to increase the flexibility of the model is to use datasets from a wide range of domains. However, creating a dataset is an expensive process because all human joints in an image must be labelled manually. Therefore, there are only limited images in a specific real-world dataset. Also, when datasets are analyzed carefully, locations of joints in different images may differ slightly depending on the judgement of the person who labelled it. In some cases, due to occlusion and lighting, human eye cannot locate position of each joint in an image. Due to these problems, pose estimation models trained on datasets loses accuracy and performance when tested on different conditions.

1.2. Literature Survey

Since pose estimation is a highly complex computer vision problem, machine learning and especially deep learning models are used in this field. General procedure is to use a pose estimation dataset to train the model. Datasets consist of labeled images. Labels are pixel locations of the keypoints in the image. Keypoints are specific joints in the body such as shoulders, ankles etc. Every dataset has its own set of keypoints. However, common keypoints used in most of the datasets are given Table 1.

Table 1: Name of Joints Used in Datasets in Literature

No	Joint	No	Joint	No	Joint
1	Left ankle	7	Pelvis	13	Right elbow
2	Left knee	8	Chest	14	Right wrist
3	Left hip	9	Left shoulder	15	Neck
4	Right ankle	10	Left elbow	16	Head
5	Right knee	11	Left wrist		
6	Right hip	12	Right shoulder		

In 2D pose estimation datasets, pixel coordinates of each keypoint are available in datasets such as (x, y).

Full datasets are usually separated into three subsets named as training subset, validation subset and test subset. Each subset contains completely different data than other two subsets. Training subset contains most of the data (80% - 90%) in the dataset. Validation data contains less data, and it is used for measuring the performance of model after each epoch during training.

Some of the datasets contain extra information. For example, COCO dataset [5] provides human segmentation information by labelling pixels as 1 and 0 and H36M dataset [6] provides camera properties such as external parameters (angle and location) and internal parameters (focal length). This dataset provides both 2D and 3D pose estimation data.

In datasets containing real human images, labels are created via two methods. First method is to label each joint manually in the image via a software. Examples for this approach are MPII and LSPet datasets. Second method is to place markers on human body around joints and later, performing image processing on taken images to find pixel location of each marker. H36M dataset [6] is the most common example of this approach.

In this part of the chapter, brief information about the datasets in the literature have been provided. In the next part, details about most common datasets have been explained.

1.3. Human Pose Estimation Datasets

Various pose estimation datasets are available in literature. Each dataset has its own different images and own different annotation. Properties of some of the common datasets are explained, and sample images are provided in the next chapter.

1.3.1. COCO Dataset

COCO is a large-scale universal computer vision dataset. It has so many useful features to be used in different parts of the computer vision fields. For example, there are 1.5 million object instances, 80 categories of objects and 250000 people in this dataset. Keypoint locations and segmentation data for people are available in this dataset and these two parameters are used as annotation for human pose estimation applications. A sample image COCO dataset showing person segmentation and keypoint locations is given in Figure 1.



Figure 1: Sample Annotated Image from COCO Dataset [7]

1.3.2. MPII Dataset

It includes approximately 25000 images containing over 40000 people with annotated body parts. Images are collected from YouTube videos containing various daily human activities. Sample images from MPII dataset is given in Figure 2.



Figure 2: Sample Images from MPII Dataset [4]

1.3.3. H36M Dataset

It includes 3.6 million 3D human poses and corresponding images. All images were taken in a large room, there are 11 actors performing 17 different activities such as discussion, smoking, talking on the phone etc. There are markers on the actors around keypoints such as shoulders, elbows and data is collected by 4 synchronized cameras at 50 Hz. Sample images from this dataset is given in Figure 3.



Figure 3: Sample Images from H36M Dataset [6]

1.3.4. CrowdPose Dataset

CrowdPose dataset [8] contains about 20000 images and a total of 80000 human poses with 14 labeled keypoints. The test set includes 8000 images. Dataset are created from crowd images extracted from MSCOCO, MPII, and AI Challenger dataset. Sample image from this dataset is given in Figure 4.



Figure 4: Sample Images from Crowdpose Dataset [8]

1.3.5. LSP/MPII-MPHB Dataset

LSP/MPII-MPHB dataset [9] includes 26675 images and 29732 humans, it is created by images from SLP and MPII datasets. Sample images from the dataset is given in Figure 5.



Figure 5: Sample Images from LSP/MPII MPHB Dataset [9]

1.4. Problems with Pose Estimation Datasets

As shown in the previous chapter, all datasets need manual labelling on images or a camera setup that creates labels automatically. Both approaches may result in errors due to reasons below:

- Manual labelling of images may be performed by different people which leads to inconsistency throughout the images
- Exact location of joints may not be achieved, especially joints occluded by other objects can be difficult to locate
- There can be human errors when labelling manually, and it is hard to check every joint in thousands of images in a large dataset.
- Physical setup to label joints automatically (like in H36M dataset) leads to error because markers can only be attached to surface of the human body but not attached to actual joints inside body. This situation leads to error when taking images of the marker from different angles.

1.5. Synthetic Datasets

Creating a pose estimation dataset can be time consuming and may lead to erroneous results due to reasons explained in the previous chapter. However, to be able to create pose estimation models that work in so many different domains, wide range of image data is needed. In that case, creating a rich and diverse dataset is a difficult problem.

To be able to solve this problem, synthetic data can be used to create dataset. Synthetic data can be the kind of data acquired from computer simulation programs. Pose estimation datasets mainly focus on human images. Therefore, different human models can be created in a simulation program, and each human model can be rigged to have different poses to create a rich dataset. Rigging is placing a skeleton system inside a human model and attaching it human body. This skeleton can be moved, and human body will follow the movement because skeleton and human body were combined and move as one part. Later, human model image and exact position of each body joint in the skeleton can be extracted from software whether it is occluded or not. Therefore, creating a synthetic dataset is an automated process which provides both images with different person instances and annotation files. Also, with additional small effort, content of the dataset can be improved via software without needing physical setup.

There are different 3D simulation programs to create human models and rig them. The most common programs can be listed as below:

- Blender
- Unity
- Unreal
- Maya
- Cinema 4D
- Zbrush

Sample human model images from each 3D program are given in Table 2.

Table 2: Sample Images from Different 3D Design Software

 Blender Human Model [10]	 Unity Human Model [11]
 Unreal Human Model [12]	 Maya Human Model [13]
 Cinema 4D Human Model [14]	 Zbrush Human Model [15]

As shown in Table 2, human models in different 3D design software are quite similar to real human images. This provides an opportunity to create a dataset from simulation program because human models in program looks like a real human and they can be used for training the model. Data created by a simulation program is called synthetic data.

Possible advantages for using synthetic data for creation of dataset is listed below:

- after setting up a system to create human models automatically, it takes a little time to create extra data
- Character joints can be acquired automatically from software without error
- size, weight, clothing, and instant pose of the character can easily be changed
- Environment, lighting, background, and foreground objects can easily be changed
- number of people and their interaction with each other
- different scenarios for occlusion can be created
- Real and accurate of joint positions can be obtained

1.6. Synthetic Datasets in the Literature

There are multiple synthetic datasets studies in the literature. SurReal [16], JTA [17] and AGORA [18] are the most common ones. In this section, features of these datasets and comparison with our dataset are provided.

1.6.1. SurReal Dataset

SurReal [16] is synthetic dataset that is created in 3D Blender software and it contains 5.5 Million images. They used 3D human models inside Blender software and replaced background with random images from LSUN dataset. To our knowledge, this is most similar study to our work. However, our work differs from Surreal dataset and differences are summarized in Table 3. Also, sample images from this dataset given in Figure 6.

Table 3: Comparison of Surreal and Metupose Dataset

Parameter	Surreal Datasets	Metupose Dataset
Person number	Single person	Single and multi person
Background environment	Only indoor	Both indoor, outdoor
Occlusion	No occlusion	Occlusion by both objects and human models
Background image	From LSUN dataset, contains humans	From Youtube or phone camera. Customizable, extendable, does not contain humans
Extendable dataset	Extendable but requires Blender and SMPL data	Blender can be used but is not a necessity. Dataset can be extended via Python.



Figure 6: Sample Images from Surreal Dataset

1.6.2. JTA Dataset

JTA [17] is a synthetic dataset created from GTA5 computer games. It is a multi person dataset and contains 10 million human objects total. Although 10 million is a huge number, most of the data comes from similar scenes. In this dataset, there are 512 different scenes, each scene has 30 seconds of video, and each video contains average 21 people. Video frame rate is 30 fps and when these numbers are multiplied; 10 million human pose is obtained.

As can be seen from here, there are only 512 scenes and some of the scenes may be redundant if human models perform regular activities such as walking instead of more complex activities. Our work differs from JTA dataset and differences are summarized in Table 4. Also, sample images from this dataset are given in Figure 7.

Table 4: Comparison of JTA and Metupose Dataset

Parameter	JTA Dataset	Metupose dataset
Scene variability	Only 512 different scenes, limited by environments in GTA 5 game	Currently 57000 background image and easily customizable and extendable
Environment type	Only urban scenarios, no indoor	Both indoor and outdoor scenarios
Extendibility	Not extendable	Extendable
Camera mode	Images only from Surveillance mode	Images from different angles



Figure 7: Sample Images from JTA Dataset

1.6.3. AGORA Dataset

AGORA [18] is a realistic synthetic dataset that contains 17000 images with 170000 static poses. These images contain 350 different human model with 4240 total human instances. Human models are 3D scanned version of real people and these models were purchased from commercial suppliers. To our knowledge AGORA is the most realistic synthetic pose estimation dataset. Our work differs from AGORA dataset and differences are summarized in Table 5. Also, sample image from this dataset is given in Figure 8.

Table 5: Comparison of AGORA and Metupose Dataset

Parameter	AGORA	Metupose Dataset
Number of person	Only multi-person	Both single and multi person
Environment	No indoor images	Both indoor and outdoor images
Camera distance	Only images from distant view angle	Images from close and distant view angle
Cost	Expensive, 3D models need to be purchased, and they are not available in final dataset	No cost, 3D models are available
Complex poses	No complex poses	Complex poses available
Variable clothing	No	Yes
Freely available	Yes	Yes



Figure 8: Sample Images from AGORA Dataset

During the time our study started, Surreal and JTA datasets were available and their performance on real applications encouraged us to move forward in synthetic dataset creation subject. Although there were multiple synthetic datasets in the literature, still there was no easy and flawless solution. Each dataset has their pros and cons and that shaped our motivation to create an easy to use, extendable dataset with various domains.

1.7. Motivation

There are multiple ways to track human movements and pose estimation is one of these methods. Pose estimation models are created by training a deep learning model with datasets. These datasets include human images and pixel location of human joints in the images. The models are trained on these datasets and trained models are used in real life situations later.

Real life performance of pose estimation models depends on two factors: Quality of dataset and quality of model used. Quality of dataset depends on multiple factors such as content of images, errors in labelling joints. Also, creating and labelling a dataset takes so much time and most of the datasets in the literature are fixed and are not increased.

To build a highly accurate pose estimation model to be used in human tracking studies, we focused on developing a high-quality dataset instead of developing model. To build dataset, synthetic data from 3D design software were used. Synthetic datasets can solve real world pose estimation dataset problems because infinitely many images can be created without any errors in labelling. Therefore, synthetic datasets are easily expandable when needed within a short duration.

Our motivation in this study is to create a synthetic dataset from 3D design software and combine this dataset with other datasets in the literature to train a pose estimation model and observe whether model's performance will increase or not.

Our second motivation is to provide 3D human models and related source code so that users can create their own dataset in the future. Also, this source code will have tools for human pose complexity analysis, person cropping and background image generation etc. In this study, we propose a synthetic pose estimation dataset and source code for dataset generation. Also, performance of models trained on these datasets were compared with other pose estimation models in the literature. Our trained pose estimation model was integrated into Jetson Nano and Raspberry Pi camera system to be used in human tracking applications.

1.8. Layout of Thesis

In chapter 1, purpose and general overview of the study was introduced. Also, literature survey about pose estimation and problems in pose estimation were provided. Our solution to these problems and contributions were provided in motivation chapter.

In chapter 2, initial studies about pose estimation were shared. In these initial studies sample images from 3 different data groups were collected and performance of pose estimation models in the literature were evaluated on these images. Image groups are:

1. Basic Human Models from V-rep software
2. 3D design software human images found from internet
3. Real world images taken from tv series

The purpose of initial study is to compare performance of models in the literature on real world data and synthetic data before building an actual synthetic dataset. Our thesis was that if a model trained on real world dataset performs well on synthetic images, then a model trained on synthetic dataset can perform well on real worlds images. This condition was observed in chapter 2. Also, concept of domain randomization was introduced.

In chapter 3, general information about Blender, which is the selected 3D design software, was introduced. Also, iterations in the process of dataset generation and different methods to create data were explained.

In chapter 4, details about used pose estimation models were introduced. Also, performance of our synthetic dataset has been evaluated using different datasets and different pose estimation models.

Finally, conclusion and recommendations for future work are presented in chapter 5.

CHAPTER 2

2. PERFORMANCE of POSE ESTIMATION MODELS on SYNTHETIC IMAGES

In chapter 1, importance of dataset to train a model has been explained and general problems with datasets have been introduced. These problems were limited number of images, human errors during labelling and amount of time spent on labelling. Synthetic datasets seem to overcome these problems because creating data is an automated process and there can be no errors with labelling. However, before building a synthetic dataset with wide range of domains, we wanted to test whether this approach will work. To be able to do that, we preferred to compare performances of real world pose estimation models on two datasets which are real world dataset and synthetic dataset. If performance of models in both datasets become similar, that will mean pose estimation models does not differentiate synthetic image data from real image data and that will lead us to move forward. Our thesis was if real world pose estimation models perform equally well on synthetic datasets, synthetic pose estimation models (models trained only with synthetic data) can perform well on real world images. This chapter covers this issue and at the end, domain randomization concept was introduced. This concept states if a model is trained only on synthetic data, with so many different conditions, then the model will perform well on real world data because it will consider real world data as another different condition like in training data.

2.1. Vrep 3D Software Information

In initial studies, synthetic data was created to evaluate the performance of real world pose estimation models. This initial synthetic data was created in V-REP software. Vrep is an open-source software to create basic simulation environment. In this environment, there

are various objects such as:

- human models
- various robotic arms
- various mechanisms
- LIDAR sensors, proximity sensors
- cameras
- cars
- basic objects like table, chair, computer, cubes
- joints, force/torque sensors
- lights

VREP is not a 3D design software, objects are imported or called into the program.

Human model was imported in VREP and moved in trajectory on a surface to get images from different angles. Sample images from this work are given in Figure 9.

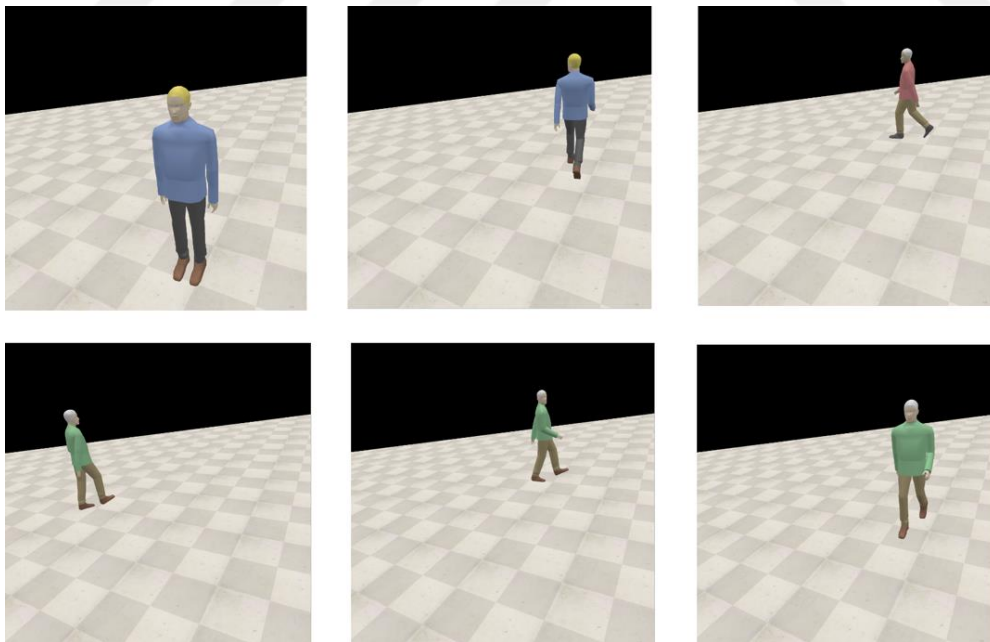


Figure 9: Human Models in Vrep Software

As shown in images above, VREP human models are not very realistic. In this software, shirt color, pants color, hair color and walking trajectory was changed during acquisition of images to create variety of situations.

2.2. Performing Pose Estimation on Different Image Groups

Total 90 VREP human model images were tested on 3 different pose estimation models. The purpose of performing pose estimation is to compare results with real world results to evaluate the behavior of pose estimation model in synthetic dataset. Sample output images are given in Figure 10.

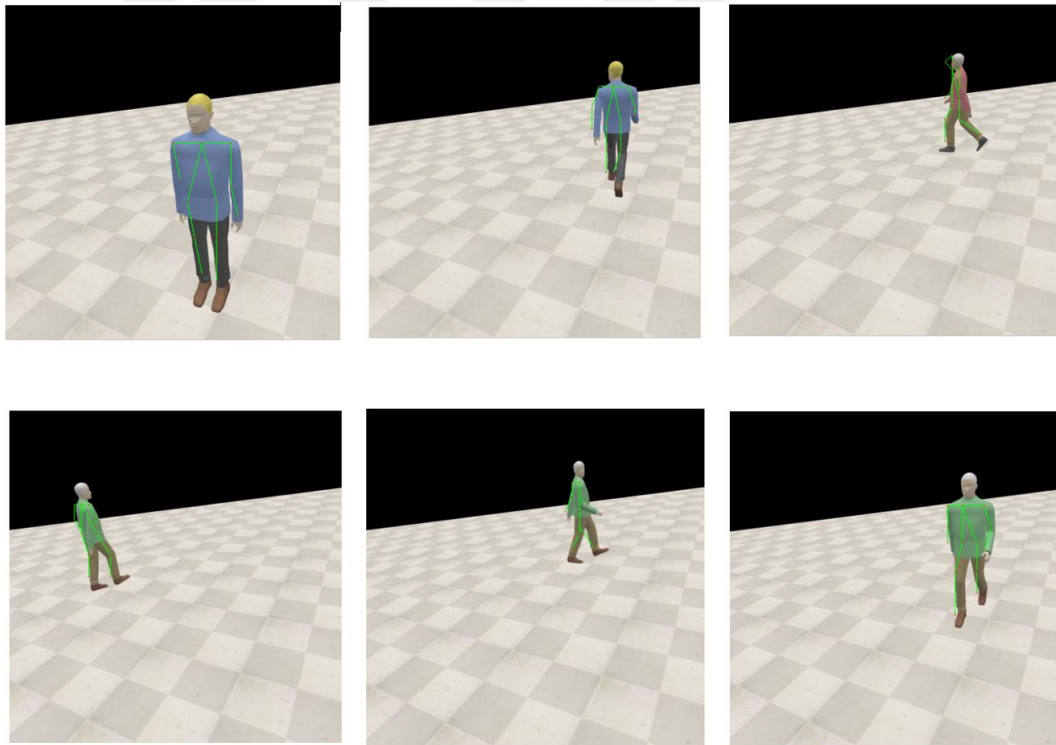


Figure 10: Outputs of Github Pose Estimation Model on images from Vrep software

After performing pose estimation on V-REP images, more realistic human models were found from internet to evaluate pose estimation models. Human models from GTA5 game and Unity software were collected. Pose estimation outputs are given in Figure 11 and Figure 12.

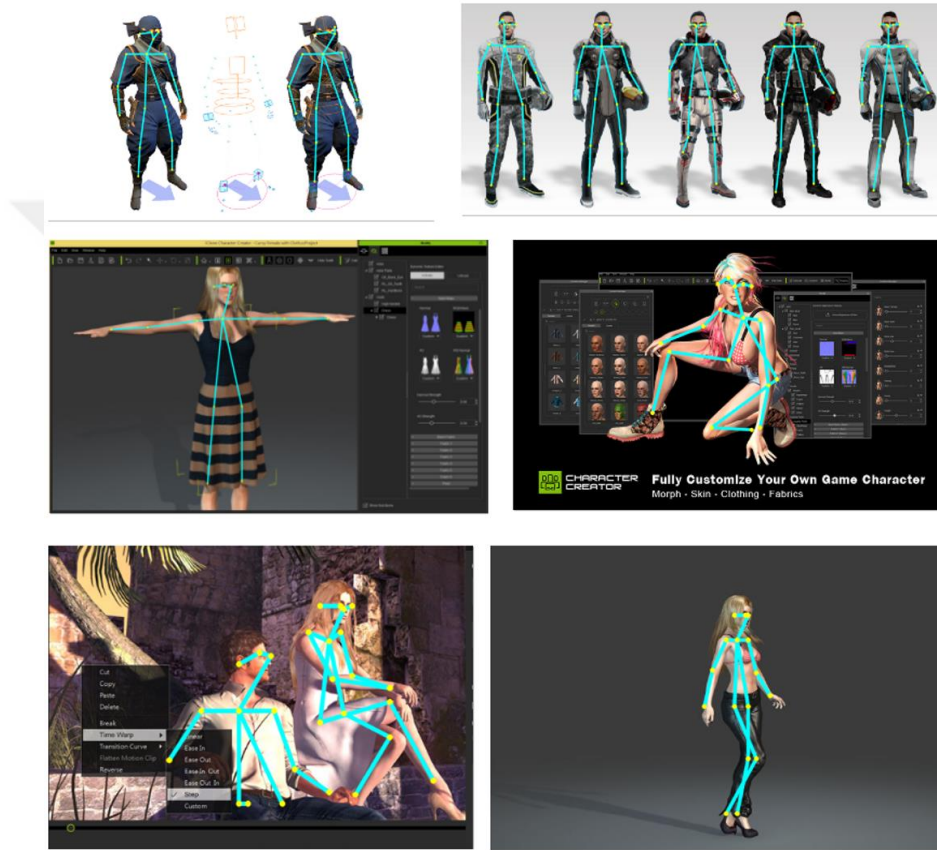


Figure 11: Pose Estimation Model Outputs on Unity Human Model Images

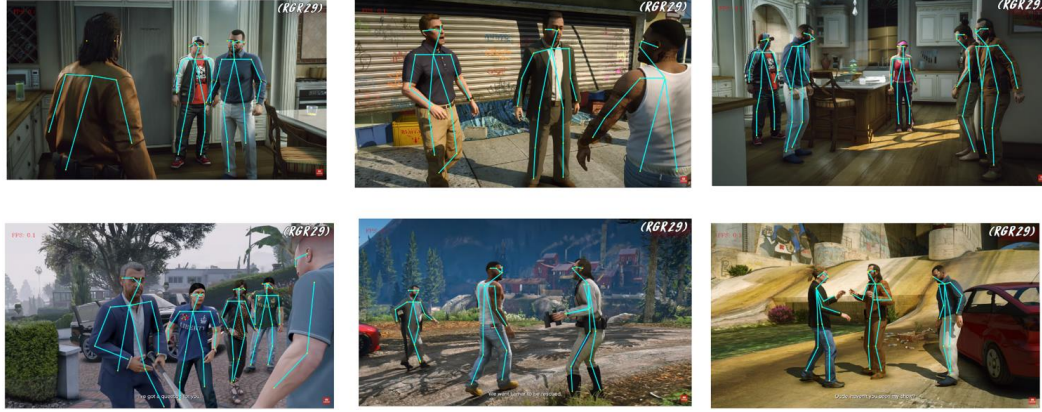


Figure 12: Pose Estimation Model Outputs on GTA5 Images

After applying pose estimation models on VREP, Unity and GTA5 human models, same pose estimation models were used on real human images. The reason is to compare results of simulation data and real data. Results for real human images are given in Figure 13.

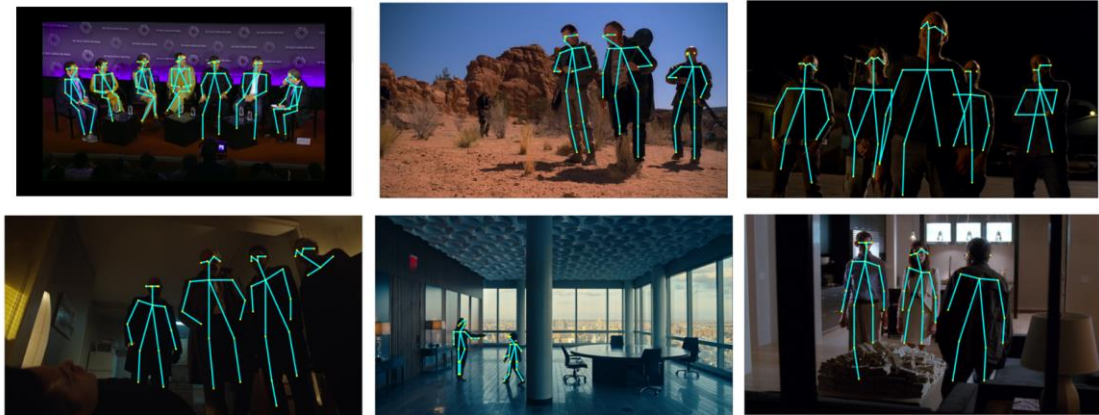


Figure 13: Pose Estimation Outputs for Real Human Images from TV Shows

So far, pose estimation models have been applied on human images from 4 different domains. These are:

- VREP software (90 images)
- Unity 3D software (45 images)

- GTA5 game (81 images)
- Real human images (100 images)

2.3. Quantifying Performances of Pose Estimation Models

Results of the same pose estimation models have been compared on 4 different domains. Because these images do not have any ground truth pose data, comparison have been performed by observation. To evaluate the performance of a model, normally, model must be evaluated on real world and synthetic datasets with ground truth data. However, this was the first application about pose estimation in thesis study, and evaluation has been made manually by observation.

Each person in images have been evaluated and one of the 5 different labels have been given in each operation. These 5 grades are:

- Perfect
- Fair
- Bad
- No Detection
- Wrong Detection (Extra person detection)

2.3.1. Perfect Label

Perfect label is given to people in the images with a very good pose estimation performance. There must be no missing joints and each detected point should be close enough to respective joint. Sample image for perfect label is given in Table 6.

2.3.2. Fair Label

Fair label was given to people in the images with almost perfect pose estimation performance. There can be one or two missing joints in a person to be considered as fair.

Sample image for fair label is given in Table 6.

2.3.3. Bad Label

Bad label was given to people in the images with worse pose estimation performance than Fair label. In this group, there must be a detection but number of missing or misplaced joints can be three or more. Sample image for Bad label is given in Table 6.

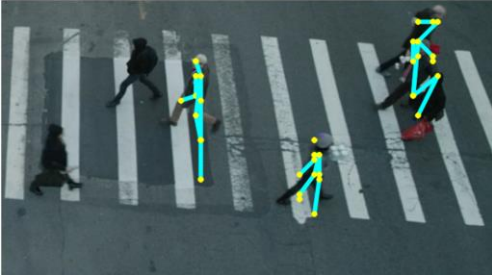
2.3.4. No Detection Label

No detection label was used when a person appears on images but pose estimation model could not find any of the joints belonging to that person. Sample image for no detection label is given in Table 6.

2.3.5. Wrong Detection (Extra person detection)

Wrong detection label was used when pose estimation model detect some joints although there was no human in detected part of the image. Sample image for wrong detection label is given in Table 6.

Table 6: Sample Images for Labels Used to Evaluate Outputs

Label Name	Sample Image
Perfect	
Fair	
Bad	
No Detection	
Wrong Detection	

Evaluation of outputs of pose estimation models in 4 different data group (real, GTA5, Unity and VREP) has been performed manually based on the criteria of labels (perfect,

fair, bad, no detection, extra detection) above. 4 different pose estimation models available in Github were used. Name of these works are given below:

Model 1: Lightweigh Human Pose Estimation 3d Demo Pytorch [19]

Model 2: Lifting from the Deep Release [20]

Model 3: Yet Another Openpose Implementation [21]

Model 4: Human Pose Estimation Opencv [22]

Performance comparison of these models are given in Table 7 to Table 10.

Table 7: Comparison of Three Different Pose Estimation Models on Real Data Group by Percentage

REAL					
REAL (338 people)	Perfect	Fair	Bad	No Detection	Wrong Detection (extra)
Model1 (%)	79	14	2	5	1
Model2 (%)	29	10	13	48	1
Model3 (%)	26	20	35	19	0

Table 8: Comparison of Three Different Pose Estimation Models on GTA5 Data Group by Percentage

GTA5					
GTA5 (165 people)	Perfect	Fair	Bad	No Detection	Wrong Detection (extra)
Model1 (%)	79	12	6	3	5
Model2 (%)	53	15	8	24	3
Model3 (%)	20	44	25	11	1

Table 9: Comparison of Three Different Pose Estimation Models on Unity Data Group by Percentage

UNITY					
UNITY (107 people)	Perfect	Fair	Bad	No Detection	Wrong Detection (extra)
Model1 (%)	77	19	2	2	0
Model2 (%)	31	31	18	20	1
Model3 (%)	24	18	36	22	0

Table 10: Comparison of Three Different Pose Estimation Models on Vrep Data Group by Percentage

V-REP					
V-REP (90 People)	Perfect	Fair	Bad	No Detection	Wrong Detection (extra)
Model1 (%)	42	50	2	6	0
Model2 (%)					
Model3 (%)	42	48	2	8	0
Model4 (%)	29	66	5	0	0

From the accuracy data given in Table 7 to 10, it can be concluded that accuracy of models in real data group is very close to accuracy of GTA5 and Unity data group. It was observed that Model 1 has the highest accuracy in all data groups so, it can give the most reliable information about detection accuracy throughout data groups. Accuracy comparison of different data group on model 1 is given in Table 11.

Table 11: Accuracy of Model 1 Throughout Different Data Groups by Percentage

MODEL 1					
Data Groups	Perfect	Fair	Bad	No Detection	Wrong Detection (extra)
Real (%)	79	14	2	5	1
GTA5 (%)	79	12	6	3	5
Unity (%)	77	19	2	2	0
VREP (%)	42	50	2	6	0

From Table 7 to 11, it can be observed that performance of different pose estimation models is consistent within Real data group, Unity data group and GTA5 data group. From these data, it can be concluded that pose estimation models trained on real data can work both on real data and synthetic data. Therefore, our thesis is the reverse of that statement. If one trains a model purely on synthetic data, this trained model may work well on real images. These results lead us to create a realistic and complex synthetic dataset from a 3D design software.

Simulation dataset can be changed, modified, labeled easily and accurately in a very short time. Therefore, if a model trained on simulation data works on real data, users can create their own data through a script without needing a physical equipment.

According to Table 11, accuracy of model 1 on VREP data group is lower and the most probable reason is that VREP human models does not contain enough details and texturing for feature detection. For example, arm and shirt has the same plain color.

Although VREP images were not helpful in comparison with real images, it was an important step to start working on a 3D simulation program. VREP studies were followed by Blender Studies. During that time following items have been achieved in VREP program:

- performing human walk on modified trajectory
- getting coordinates of every joint during walking through Python VREP API
- Modifying hair color of human model

- Modifying shirt and pants color
- Adding objects to simulation

These items were important learning blocks to start working on Blender which is a complex 3D design software for creating synthetic dataset.

2.4. Domain Randomization

In machine learning, domain randomization is an important concept. Basically, when a model is trained on a wide variety of synthetic data, it can work on real world data as well because model perceives real world data as another variation of synthetic data. Examples for variation of data in simulation dataset can be listed as:

- different colors
- different relative sizes of objects
- different camera angles, multiple images of the same instance from different angles
- different lighting
- different number of objects
- different textures and shapes of objects

In a synthetic dataset, it is easy to change parameters listed above and to create new data with a wide range of features. Therefore, when a model is trained on large synthetic dataset, this model will work well on real world data because model will perceive real world data as another variation or sub-category of large synthetic dataset.

One study [23] used domain randomization to explore capabilities of a model trained only on synthetic data. They performed object localization and grasping and using a model only trained on simulation data, they achieved 1.5 cm accuracy in real world conditions model was robust enough to distractors and occlusions.

Another study [24] used domain randomization to perform car detection and by using only simulation data, they achieved performance comparable to real world dataset.

Another study [25] increased performance of domain randomization even further. They

used active domain randomization concept. Basically, this concept uses reinforcement learning to search for most informative parameters in domain randomization and uses these parameters more often when creating dataset and this leads to increase in overall accuracy because the content of the simulation dataset was optimized for maximum training accuracy.

2.5. Final Notes

When pose estimation models were tested on real images, Unity 3D software images and GTA5 images, accuracy results were consistent. That means, a model trained only on real world data can work well on simulation data as well. Also, there was a concept called domain randomization which utilizes wide variety of synthetic data on training a model and uses this model on real world data with high accuracy. Based on these facts, we created an environment on Blender 3D software to create pose estimation dataset and we used this dataset to train a pose estimation model. After training, accuracy of the pretrained model was increased. Our trained model was deployed in Nvidia Jetson Nano and Raspberry Pi camera system to be used in human tracking applications in the future.

After getting promising results explained in this chapter, creating dataset in a 3D design software have been decided and Blender software was chosen for this work. In the next chapter, capabilities of Blender software have been explained and methods and procedures to create synthetic dataset from Blender have been discussed.



CHAPTER 3

3. STUDIES ON BLENDER 3D SOFTWARE

In chapter 2, based on tests performed on real world pose estimation models, it was observed that real world models worked equally well on real world data and synthetic data. Therefore, that lead us to create realistic dataset from a 3D design software. Blender software was selected to create human models and environment.

In this chapter, general information about Blender 3D design software has been introduced and methods and procedures to create synthetic datasets have been explained. These methods include:

- environment design
- human model design
- extraction of joint locations via Python
- creating non-repeating data
- creating data from different domains
- pose complexity analysis

One of most important steps to create high quality synthetic dataset is to cover wide range of domains that can appear in real life situations. Methods used to increase the variability of data were illustrated in this chapter.

3.1. Blender General Information

Blender is the free and open-source 3D creation suite. It supports the entirety of the 3D pipeline modeling, rigging, animation, simulation, rendering, compositing and motion tracking, even video editing and game creation. It is very popular in graphical design

market and is fully supported in Windows, Linux and MacOS operating systems. Blender website has complete documentation of the software and because it is so popular there are so many Youtube tutorials, forums, and websites to learn Blender. This software comes with built-in installed Python. This python is completely inside the Blender program and has no connection with regular Python installed on computer. Blender has a library in Python called bpy. By using this library, one can change or modify so many objects inside Blender program.

Because it is completely open source, comes with preinstalled Python inside and has a python library which provides so much flexibility, Blender software was selected to create synthetic data. In the next part, Blender properties and capabilities regarding pose estimation has been explained.

3.2. Blender Properties and Capabilities Regarding Pose Estimation

3.2.1. Creating Human Models

There are multiple ways used to create realistic human models in Blender. First one is downloading human models from internet and importing them into Blender. Because Blender is an open-source free software, there are so many websites that provide free or paid Blender models, and some parts of these models are realistic human models. Sample human images from Turbosquid website are given in Figure 14.

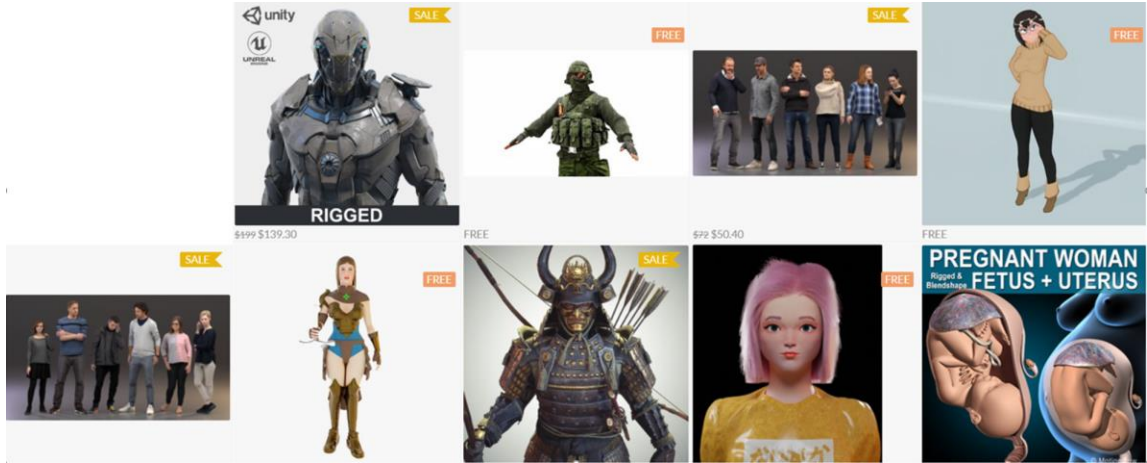


Figure 14: Blender Human Models in Turbosquid Website [26]

Second way to create human models is using add-ons. Add-ons are extra software designed for a specific purpose and they can be installed in Blender. There is an open-source add-on used to create realistic human models in Blender. This add-on is called MbLab [27]. This add-on has 14 different human models in it and when a model is created, its age, mass, hair, skin color, size and body type can be modified through a GUI. All human models in the final dataset were created by MbLab [27] add-on method. Sample image for a human model created by MbLab [27] tool is given in Figure 15.



Figure 15: Default Human Model Created by MbLab add-on in Blender

Third option is to create human models from scratch. However, creating a realistic human model requires hard work and expertise in 3D design field, and it is not an automated process. To demonstrate the process of creating a human model from scratch, sample images given in Figure 16 show the 6 steps are given below. These images were taken from YouTube Blender tutorials.

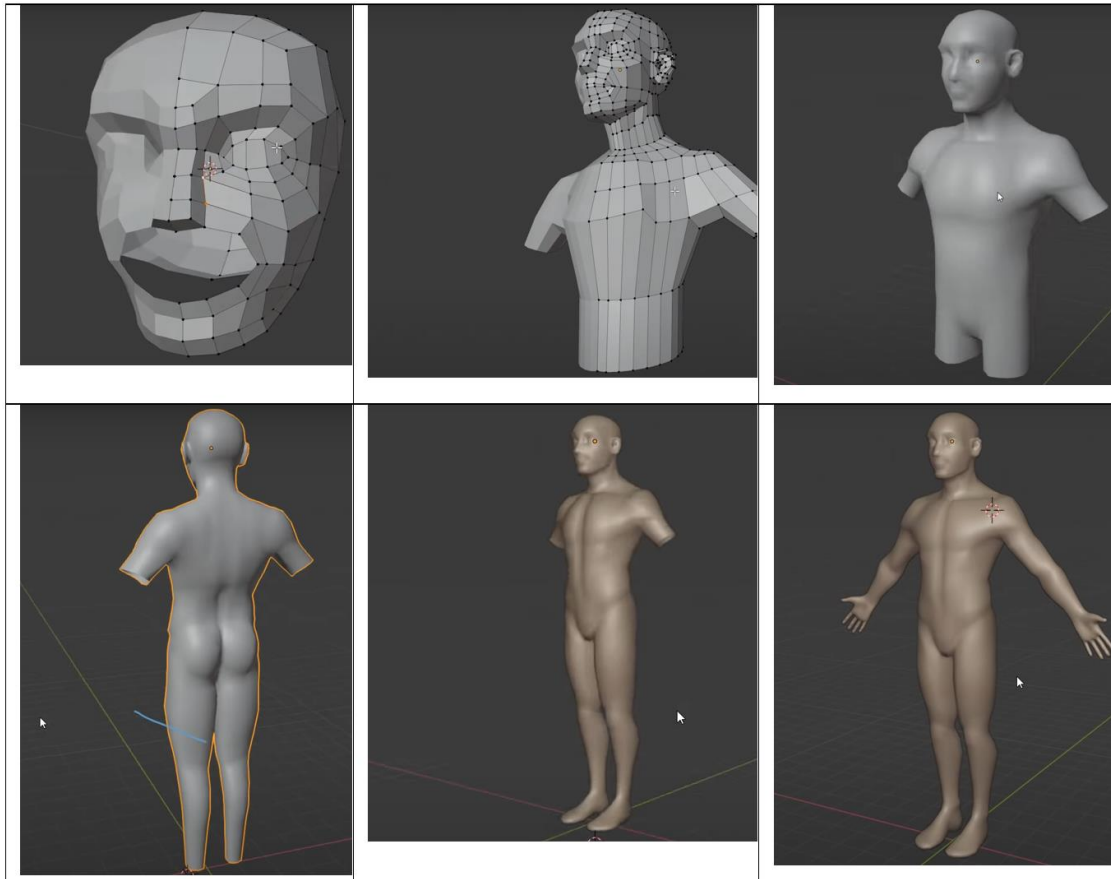


Figure 16: Demonstration of 6 Steps to Create a Human Model from Scratch

In this part of the chapter, methods used to create realistic human models were explained. Second method (MbLab addon) was selected to create human models due to ease of use and configuration options. In Blender, joint position and rotation data can only be obtained

if a human model has a skeleton system. In Blender, skeleton system moves accordingly with human body and in the next part, methods used to acquire joint data from skeleton system has been explained.

3.2.2. Person Rigging and Animation

Person rigging is integrating a skeleton system inside a human model and, when a joint in the skeleton system moves, body moves accordingly. Each movement in skeleton leads to different pose of human body. Also, joint locations at any instance can be obtained through python API. Sample image for human body rigging is given in Figure 17.

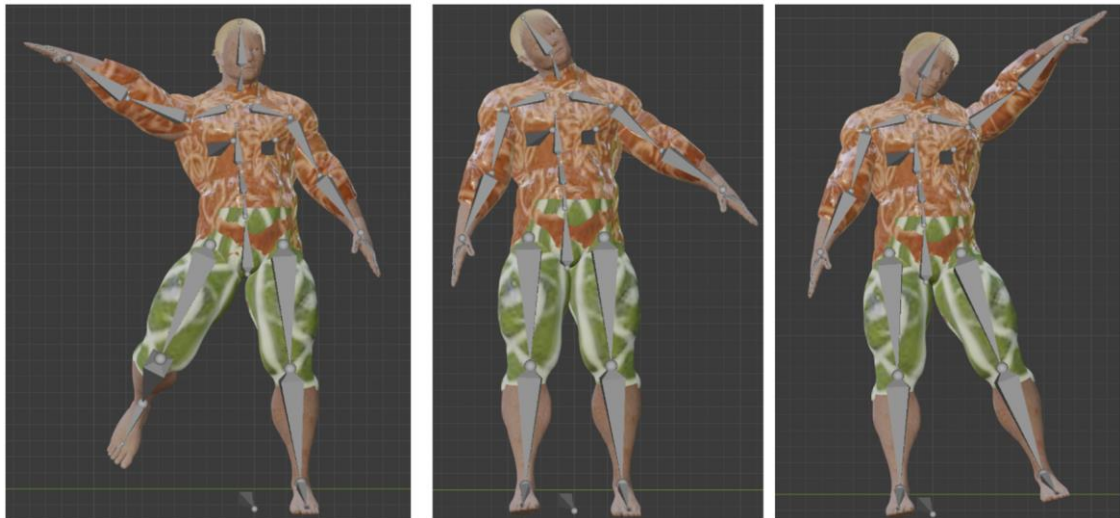


Figure 17: Three Different Poses of the Same Human Model and Joints

MbLab Blender addon was chosen to create human models and it comes with default rigging feature as illustrated in Figure 17. Also, MbLAB addon has an option for realistic joint limitations. Therefore, when a human body joint is forced to do an unrealistic joint movement, software limits that movement to stay within human joint movement limitations. If necessary, skeleton system can be built from scratch as well.

After creating a human model and finish rigging, joint data is available to collect, and this

is performed by built-in Python feature inside Blender software. Also, built-in Python enable user to change some of the parameters from script in an automated way. In the next part, details about Python API have been discussed.

3.2.3. Python API

Blender comes with built-in Python API. Python was used for changing parameters from script to increase variability of dataset and it was also used for acquiring joint data to create annotation files of dataset.

Blender python has a library called bpy and this library can change position, rotation, color, texture of an object and it can get information of object position, rotation. In this study, following parameters have been performed from Python script:

- Getting 2d and 3d joint locations of each human in each frame
- Saving joints data in a csv file
- Changing color and texture of human cloth, hair, and skin
- Changing position of human model, camera, and lights

Python script is so important to create randomized human models and environment in an automated way. In this part, capabilities of Python API to create a dataset with a wide range of domains in an automated way has been explained. In the next part, details about randomization of parameters have been discussed.

3.2.4. Changing Appearance of Human Model

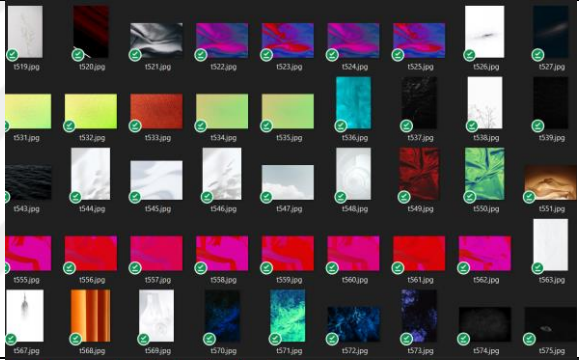
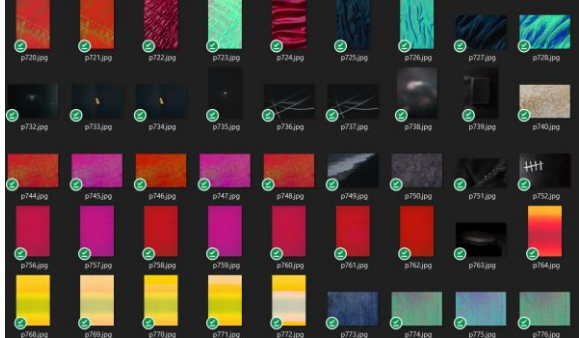
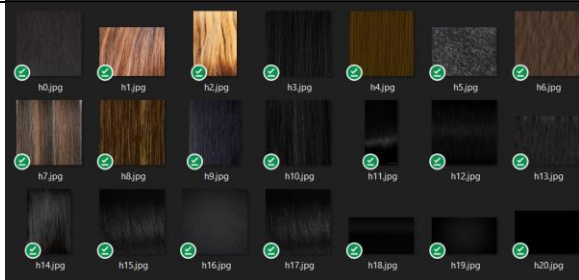
In Blender, it is possible to change texture and color of t-shirt, pants, hair and skin and this is an important step to create randomized data because if all models have the same appearance, that may lead to overfitting during training. To avoid that, these parameters were changed automatically in every frame via Python script. Different textures have been applied to each part to change appearance. Numbers of textures were given below:

- 1069 different t-shirt textures

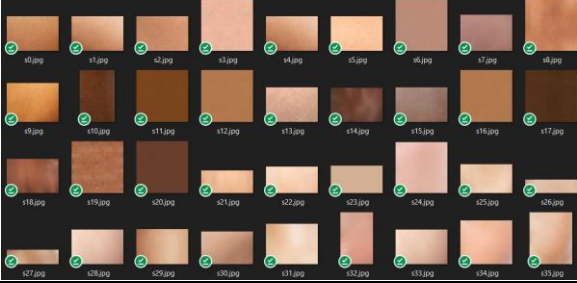
- 1069 different pants textures (same 1069 images with t-shirt textures)
- 21 different hair textures
- 36 different skin textures

These texture images were found from Google and sample images for each group are shown in Table 12.

Table 12: Sample Images for Textures in Blender

Category	Texture Images
Tshirt Textures (1069 units)	
Pants Textures (1069 units)	
Hair Textures (21 units)	

(Table 12 Cont'd)

Skin Textures (36 units)	
--------------------------	--

After selecting a random texture for t-shirt, pants, hair and skin, human model is completely ready and sample images are given in Figure 18.



Figure 18: Human models created by different textures

In this part, textures used to create various human models have been discussed. Firstly, human model was created, then rigging has been applied and finally, texturing has been applied on human model to have a different appearance in every image in the dataset. After these steps, final images to create synthetic dataset can be acquired via rendering and it has been discussed in the next part of the chapter.

3.3. Image Creation in Blender

After human models have been created and rigging and texturing have been performed, it is possible to get 2D, 3D joint locations through Python API. However, to create a robust dataset, not only human model but also environment has to be designed carefully to cover wide range of domains in the final datasets. There were three methods used to create environment. First one was to design a street that has human models, houses, trees, roads etc. After applying this method, it was observed that creating large amount of nonrepeating synthetic data was not feasible and it took too much time. Therefore, a second method was proposed. In this method, background image of the model has been changed in every frame to create variability in different images because each image in the dataset has a different background from various domains. Although the second method provided good results it still lacks features such as multi-person and occlusion so, a third method is proposed. In the third method, human model was placed in a green space in Blender and cropped from green background to paste onto new changing background later. This method allowed us to have multi person dataset with occlusion feature. Only third method (cropped person method) is used for dataset generation. Details about these three methods have been explained in the next parts.

Before getting into details about each method, details about capabilities of Blender software were provided below:

3.3.1. Creating a Street Environment

Blender has an addon to create buildings called Building Tools. Following parameters can be changed easily via this addon to create a new building:

- Number of floors
- Number of windows, doors, and balconies
- Shape of windows, doors, and balconies
- Roof type

- Height and width of building
- Location of building

Also, by changing textures in each surface, appearance of each building can be changed automatically. Also, tree models have been added. After that, human models were placed on the street and image and joint data were collected to create synthetic datasets. Images are obtained via rendering output of Blender software and resolution is 960x540. Auxiliary view of street and human models placed in the street are given in Figure 19 and Figure 20 respectively.

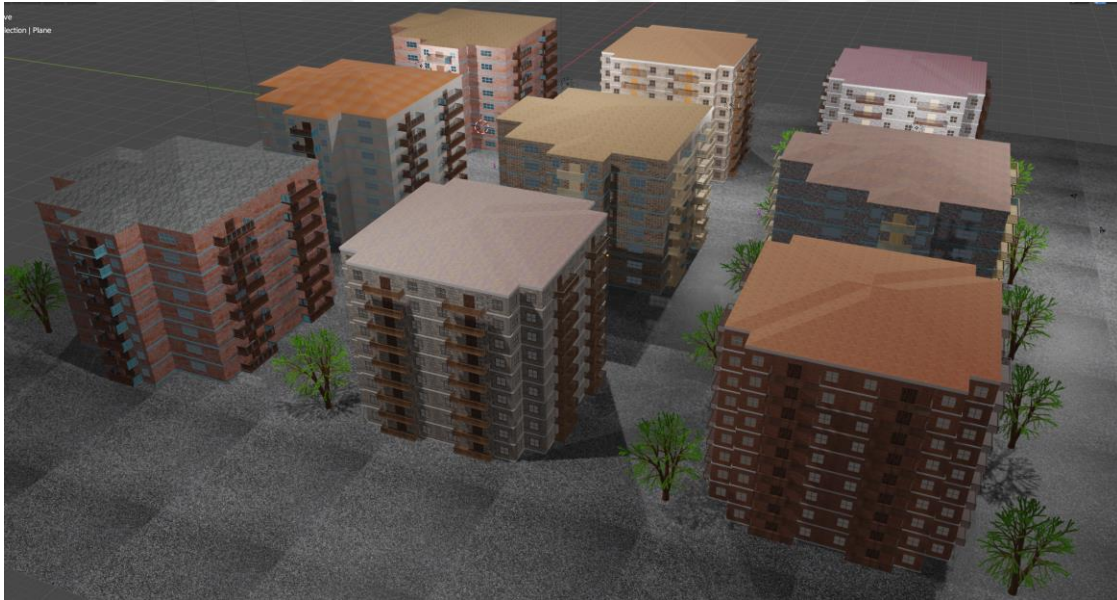


Figure 19: Sample Image for Street Model Created in Blender



Figure 20: Human Models Placed in the Street Model in Blender

Although street view method is an automated process to create synthetic dataset, it does not provide variety of objects in the environment. This may cause problems to work on different domains later because there needs to be variability not only in human model data but also in environment data to create a robust synthetic dataset. Therefore, second method was proposed which is changing background of human model. Details of this method was discussed in the next part.

3.3.2. Human Model in Front of a Plane Video Method

Although street view explained in the previous part provides so much flexibility and automation in the process of creating dataset, it was observed that it was not scalable enough which means it would take so much time to create dataset with thousands of non-repeating images containing different objects in the environment. To illustrate, outside

view of the street used to create dataset is given below. When camera goes through the entire street taking images of human models in each frame, the whole street provides only 150 images.

A sample image from street model is given in Figure 21.



Figure 21: Outside view of Street Model in Blender

Most of the real-world datasets contains more than 10000 images. Therefore, required simulation images will be even more than 10000. To solve this problem, street model, buildings and everything was completely removed, and human model was placed in front of a plane and that plane shows images like a screen. Therefore, background image of the human model was changed in every image, and this simulated environment change. Demonstration of this method is shown in Figure 22 and Figure 23.



Figure 22: Human Model in front of a Video Plane (Rendered View)

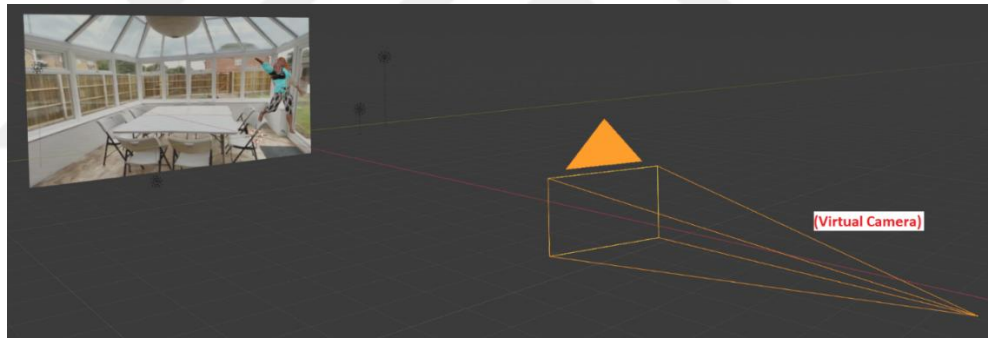


Figure 23: Human Model in front of a Video Plane (Outside View)

Human model pose, location, clothing, hair color, skin color and background image are different in every image in the dataset. Different background creates the effect of different environment domains. Example images created using this method are given in Figure 24.

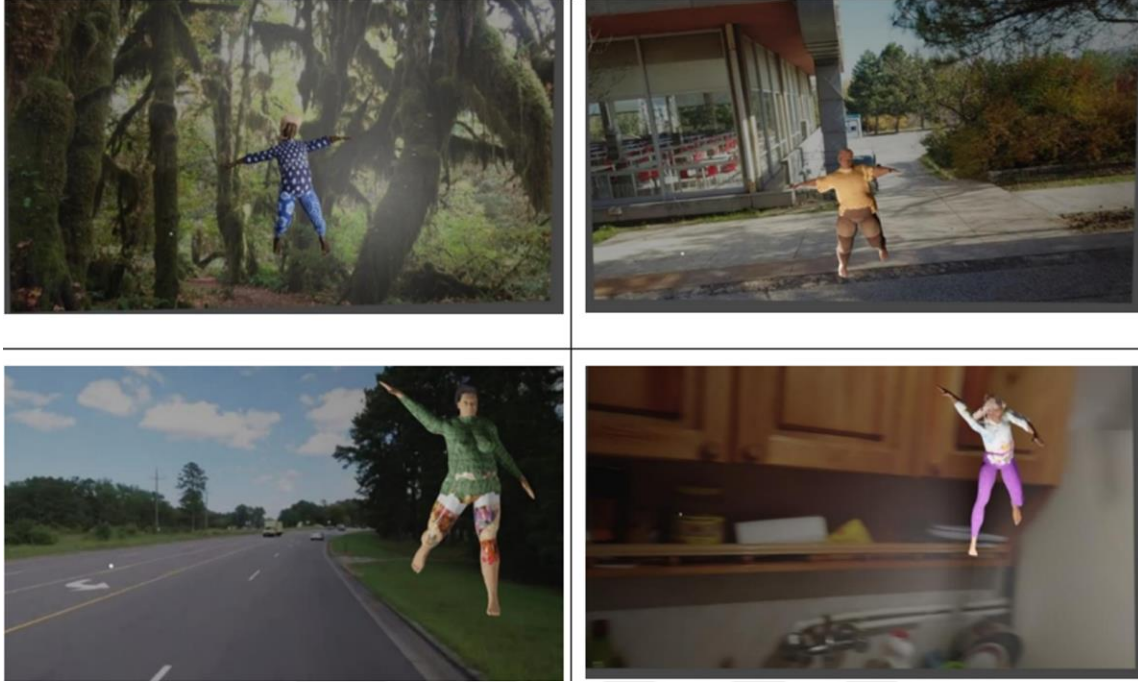


Figure 24: Sample Images with Changing Background

3.3.3. Person Cropping Method

Although the street method did not provide enough flexibility with the environment, plane video method solved these problems with changing and varying background. However, plane video method could only provide single person data without occlusion and all data generation is performed in 3D Blender software, which means if a user wants to expand Metupose dataset in the future, he needs to install Blender, install necessary addons, necessary libraries in Python which differs from regular Python and run Blender on a powerful GPU.

To provide more flexibility in dataset generation, human cropping method has been developed. In that method, human model images are taken in green colored 3D Blender space and cropped. Later, all cropped images are pasted on different background in random coordinates at random scales. Once the person images have been cropped, they

can be used in all datasets in the future by using only regular Python.

After cropping images, all operations are performed in Python and Blender software is not necessary.

Firstly, in Blender software, human model, cameras and lights were placed inside a very large, 100-meter diameter green colored sphere to create a green screen effect. The reason to use a very large sphere is that regardless of camera position and angle, background will always be green because all the objects are in a green sphere. A sample image is given in Figure 25.

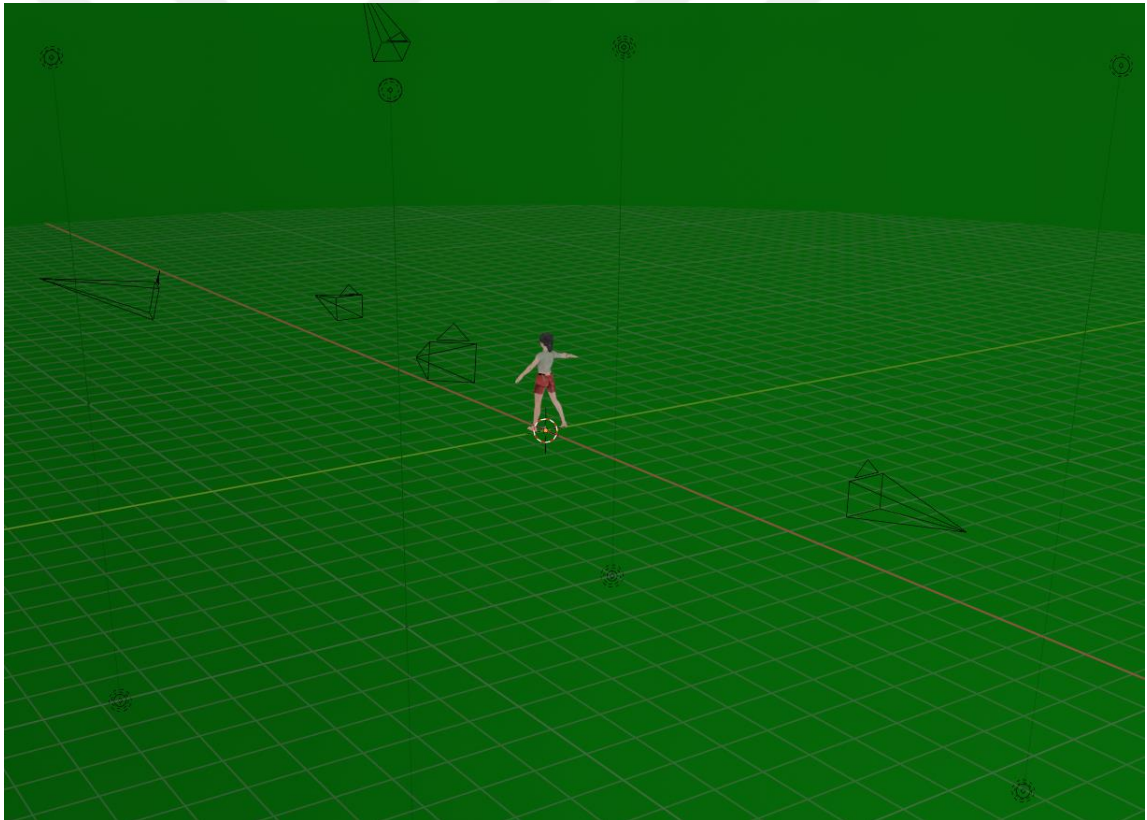


Figure 25: Human Model, Cameras and Lights inside 100 Meters Diameter Green Sphere in Blender

After setting Blender environment, human model performs different movements and images are rendered from the view angle of 5 different cameras. Sample result images with green background and their cropped version are shown in Figure 26:

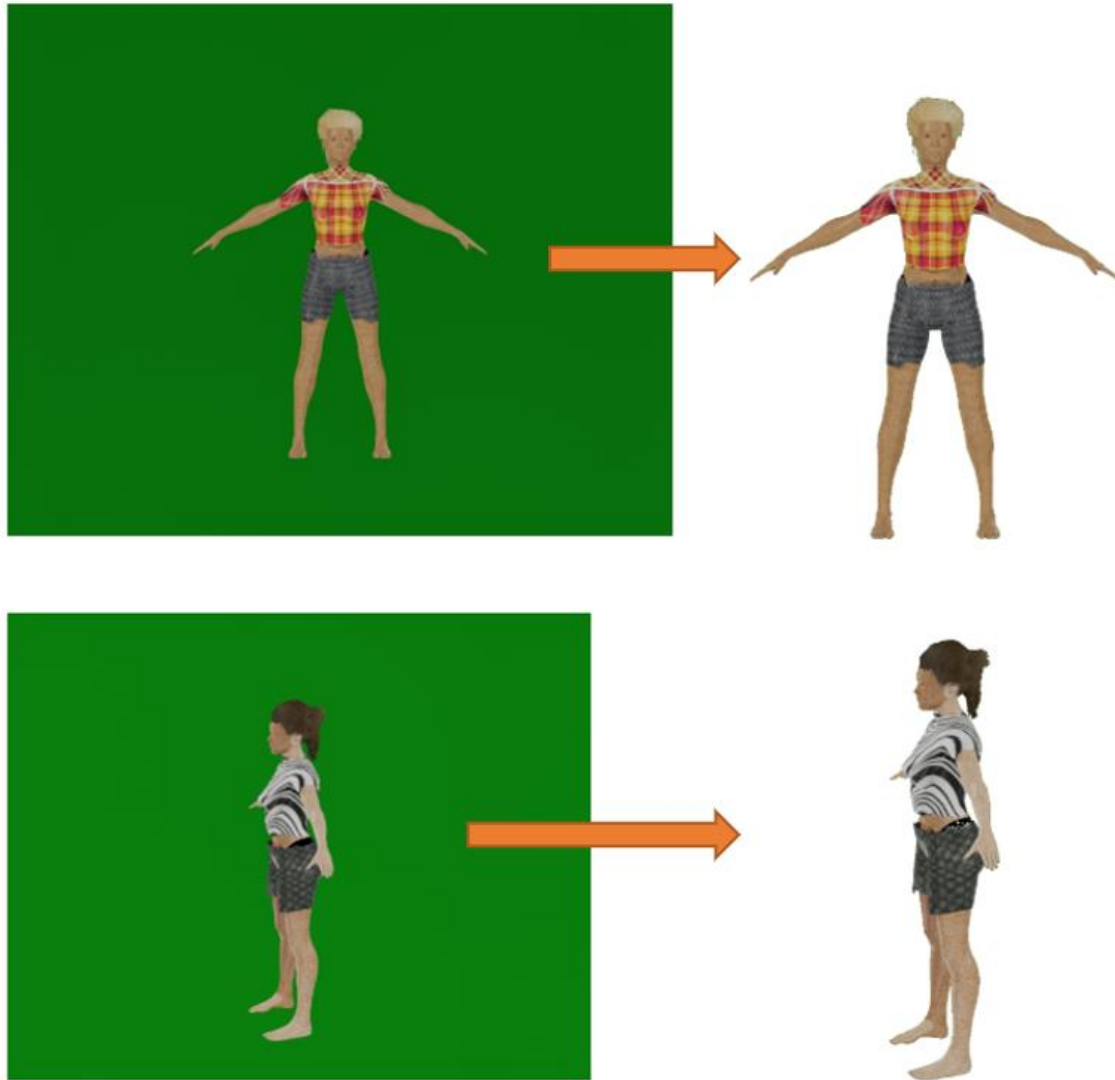


Figure 26: Left Side: Images Rendered from 3D Blender Space, Right Side: Cropped Version of Images in Opencv

Firstly, images are cropped via green color masking in opencv Python. After cropping, we have all pixel information about cropped image.

After that, object images with transparent background have been downloaded from internet. They were used as occlusion objects. From 26 different categories over 1000 object images have been downloaded. In the second method we had used plane video method and from that method, we already had 57000 different and independent images to be used as background. These background images were obtained from different YouTube videos that does not contain human in it. However, to ensure there is absolutely no person in any of the background images, we performed pose estimation on 57000 images and detected human in only small percentage of all images and deleted those images. Lightweight Human Pose Estimation 3d Demo Pytorch [19] model was used for pose estimation and to be safe, we set confidence threshold level very low to detect any person even in harder cases. We also provide source code for detecting human in video and deleting this frame. User only need to provide video itself, and number of second between each frame and code will extract frames as images with no human in it.

Firstly, cropped human models are pasted on background image as shown in Figure 27:

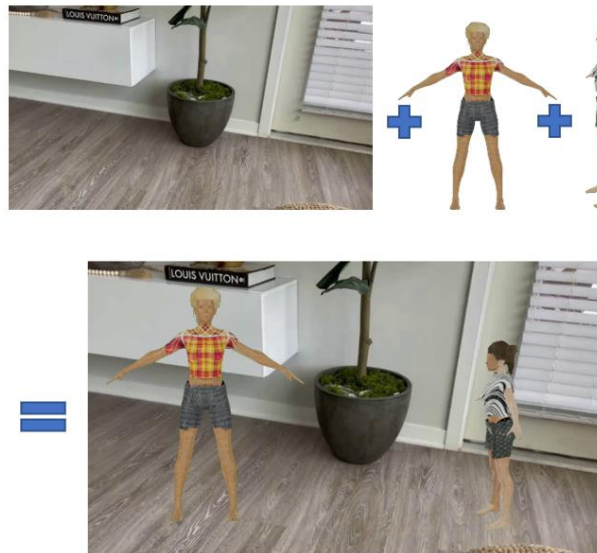


Figure 27: Cropped Human Model Images Pasted onto Variable Background

Later, occlusion objects are randomly selected from object library, scaled and pasted on image to create occlusion. Occluded person image is given in Figure 28.



Figure 28: Various Objects Added to Image to Create Occlusion

During creation of dataset following steps has been taken into consideration:

1. Content of background images (indoor, outdoor) was known before, therefore, only relevant objects suitable with the background has been used in each image. For example, tree objects are only used with outdoor background images not indoor.

2. Cropped images were not directly pasted, their scales have been randomly changed before pasting.
3. If an image is multiple person image, scale of each person is random but also close to other people in the image. For example, in an image, height of one person cannot be 5 times the height of another person. The range changes between 0.8 and 1.2.
4. Scale of occlusion objects have been set relative to human size in the image. For example, a luggage cannot be larger than any human in any image etc. and if there is a laptop, laptop has to be smaller than luggage as well. Each object has a dimension coefficient.
5. Information whether a joint is occluded or truncated is available, but occlusion information is not utilized in MPII dataset format. However, user will still have that information in our dataset.
6. We provide final, ready to use dataset and also the source code. Users can use our dataset or they can create their own dataset with our source code. Content of our source code is summarized in Table 13.

Table 13: Content of Our Source Code

No	Item	Explanation
1	27 different Blender Models	Each model can provide 5000 green screen images from 5 different camera angles
2	Textures for clothing	1000+ texture images for clothing and various textures for hair and skin color
3	49000 cropped person images	Users can use our cropped person images; they don't have to deal with Blender
4	57000 background images	Users can use our background images obtained from Youtube videos with no human in it
5	Background image creation source	Users can create their custom background images by providing a video. Our code will detect people in video and provide only frames with no human in it
6	Pose complexity analysis code	Users can scan through entire cropped person images and entire final dataset images sort images based on difficulty of poses
7	Occlusion objects	From 26 different categories, over 1000 object images with transparent background were provided to create occlusion
8	Dataset generation code	Users can create their dataset by using our code and cropped images
9	Actual final dataset	We provide final, ready to use dataset with 178000 images and 402000 people in it.

Links to the source code is given in Appendix.

As shown in Table 13, so many different contents are available in the source code for different purposes and there are different ways to create dataset. For simplicity, we wanted to separate these methods into three parts. These are:

1. GPU + CPU available option
2. Only CPU available option
3. Nothing available option

GPU + CPU Available Option

If users have a powerful GPU and CPU, they can use our Blender human models to create dataset from scratch. We provide 27 different Blender human models, and inside Blender, they can edit following parameters:

- Human model appearance
- animations, movements
- human scales
- joint content in dataset
- texture of clothing
- lighting, camera angle

To create variability, users can edit these features manually but, in fact, they don't have to. We provide 27 human models and each human model has 5 different camera view and this makes 135 configuration and each configuration provides 1000 different green screen image. Every time they run Blender even for the same human model with same camera, they will still get different hair color, skin color, clothing, and body size every time. By keeping everything same, only similarity between different runs will be type of the movement itself. However, when camera or human model is changed, everything will change as well.

After running Blender which means rendering images, users will get human model images with green screen background. Sample image is given in Figure 29.

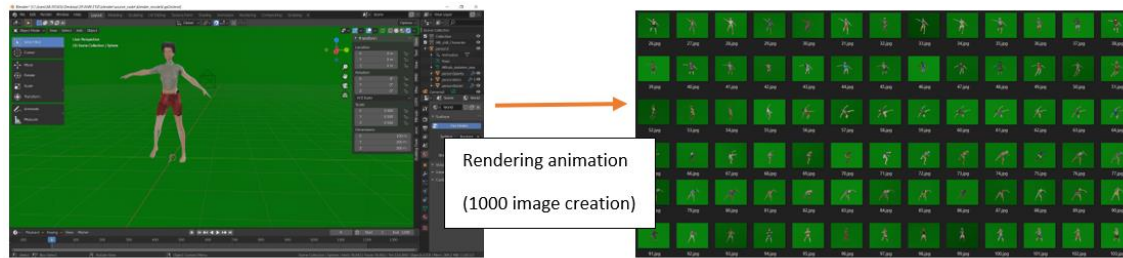


Figure 29: Rendering Images from Blender

Later, each green screen images are cropped to remove green background. A sample image is given in Figure 30.

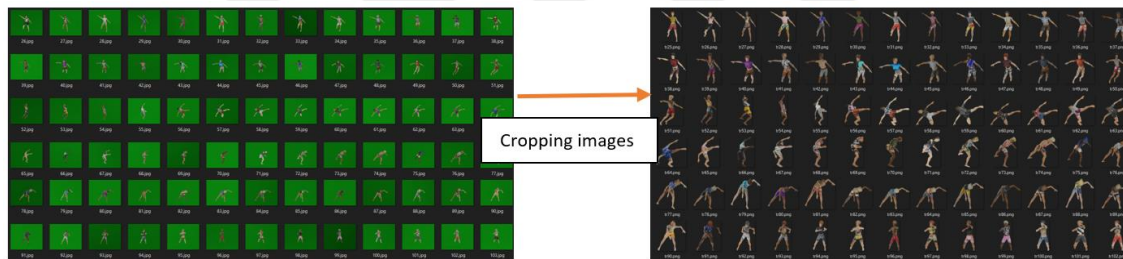


Figure 30: Cropping Human Models from Green Screen Images

Every step followed so far is one-time only process. This means the final dataset is created by selecting human models from cropped image library. Once the cropped image library has been created, infinitely many final datasets can be created randomly from these cropped images without using Blender. The reason to use Blender is to enrich the content of cropped image library which is base of synthetic dataset. We currently provide 49000 different cropped images and if users want to expand the content of that library, they can use Blender otherwise, Blender is not necessary.

After getting cropped images, they can automatically create synthetic dataset by only specifying a few parameters such as image number, occlusion level etc. All parameters will be set from config.csv file.

Only CPU Available Option

Blender can utilize both CPU and GPU but when it runs only on CPU it works slower. Therefore, this option is for the users that don't have a GPU in their computer. Users can download our 49000 cropped images and create their own dataset. Every time they create a new dataset, they will get different images because in every time, code will choose random cropped images, and paste them onto random background images alongside with random objects. For this option, only Python is necessary.

Nothing Available Option

In this option, nothing is necessary because we already provide a sample final dataset.

Users can download this dataset and use it. Our sample dataset contains:

54000 x 1 person images

54000 x 2 person images

40000 x 3 person images

30000 x 4 person images

That makes total 178000 images with 402000 people in it. For comparison, MPII dataset contains total 25000 people in it.

One of the best features of our dataset is that it is completely extendable by running our source code. Also, users can change content of the background and clothing textures for their specific purpose.

3.3.4. Pose Complexity Analysis

During creation of dataset, one of the most important elements is to have variable, unique and nonredundant dataset. To achieve that, different background image, clothing, lighting, camera angle, human size etc. was used to minimize the redundancy. However, these parameters only covered appearance of the image but not body pose itself. For example, a dataset may contain infinitely many lightings, coloring etc. but pose estimation models are actually trained on location of joint data. Therefore, we created a new method to

measure complexity of the poses in created images. Pose complexity can be applied to individual cropped images and final images as well.

For example, when we create 50000 image dataset, to avoid redundancy, we may need images with most unique and different poses in it. In our source, we can specify this feature by giving (0, 60) numbers which means only accept images that has highest scores between top 0% and top 60%. This means, after creating dataset, our pose complexity algorithm scans through entire dataset, then score each image based on its complexity and accept top 60% score and discard bottom 40% scores.

The benefit of measuring pose complexity is to discard most common and repetitive poses and only accept most unique and difficult poses to achieve maximum performance with minimum number of training images. Too many redundant and repetitive images may lead to higher storage requirements and higher training duration.

To our knowledge, there was only one study that performed pose estimation complexity method when creating dataset and it was MPII dataset. In that study, without giving equation details, it was briefly explained as, images with the most uncommon and different poses compared to rest of the dataset has the maximum pose complexity score. Therefore, by considering that statement as our starting point, we created a new pose complexity metric. In most simple terms, algorithm scans through entire dataset, extract all pose vectors (a vector from shoulder to elbow, knee to ankle etc.) then categorizes all vectors based on x and y coordinates, then gives high scores to vectors, if that vector has a unique or uncommon (x, y) vector.

For demonstration, let's consider there are 10 people in a final dataset and we work on joint vector between left shoulder and left elbow. This means we will have 10 different 2D vectors. A sample vector from one of the images is shown in Figure 31.



$\text{Vector}_{11} = (\underline{64 \text{ pixel}}, 72 \text{ pixel}) = (64, 72)$

Figure 31: Sample Image Showing a Joint Vector on a Human Model

In the example above, vector dimensions for left shoulder to elbow was obtained. For that joint, 9 other vectors will also be obtained from rest of the dataset ($10-1 = 9$ images in the dataset). After getting all 10 vectors from 10 images in the entire dataset, all of them will be placed on 2D coordinate system as shown in Figure 32.

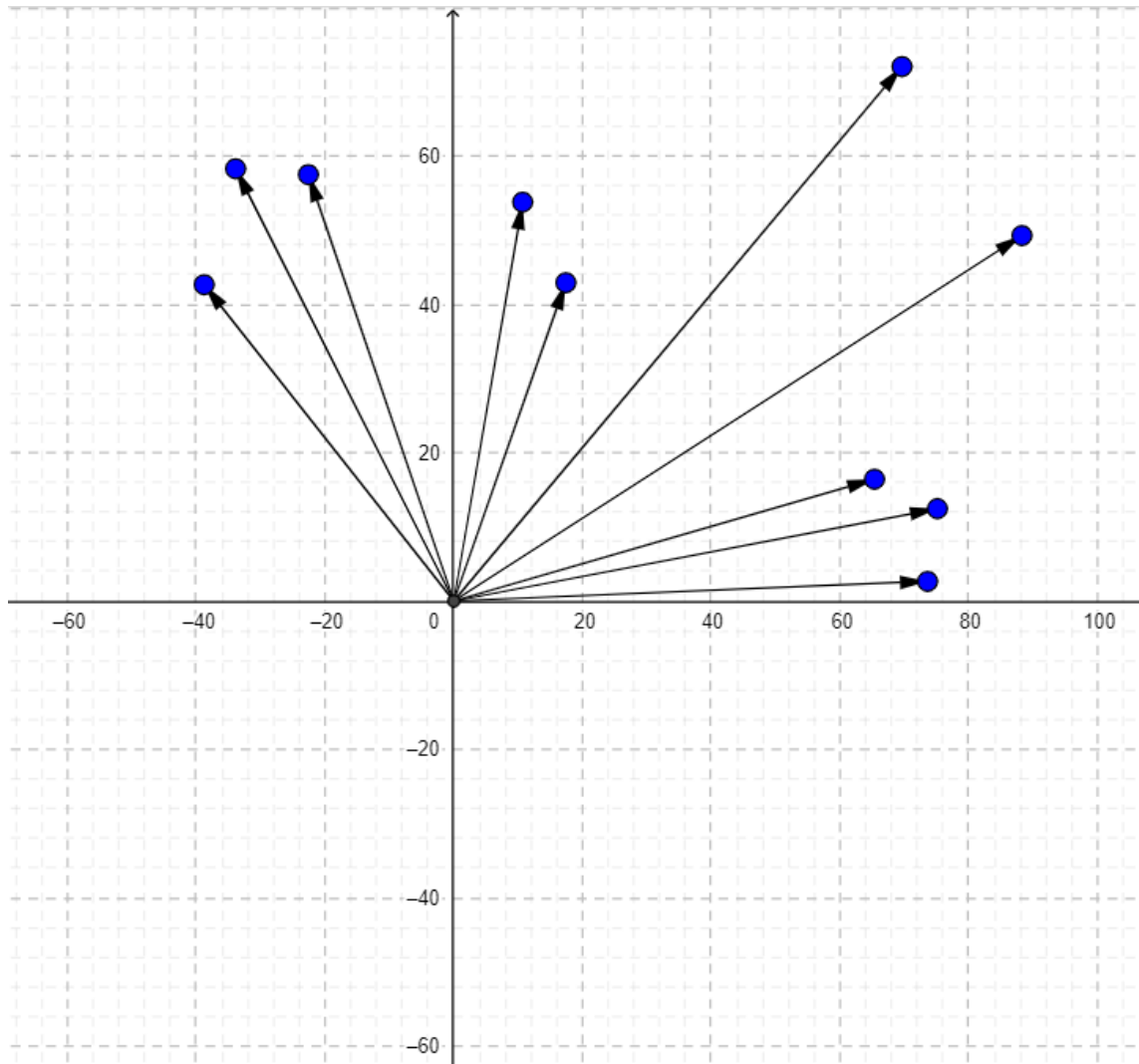


Figure 32: Joint Vectors for Left Shoulder to Elbow in Entire Dataset with 10 Images

As shown in grids, let's say we categorize entire 2D spaces with 20-pixel grids. This means, in a 2D space bounded by $(-60, 80)$ and $(100, -60)$ coordinates as shown above, by dividing it into 20 pixels grids, there will be total 56 grid squares (56 categories). In these categories, some of the poses fall into the same grid square (same category) as shown in Figure 33.

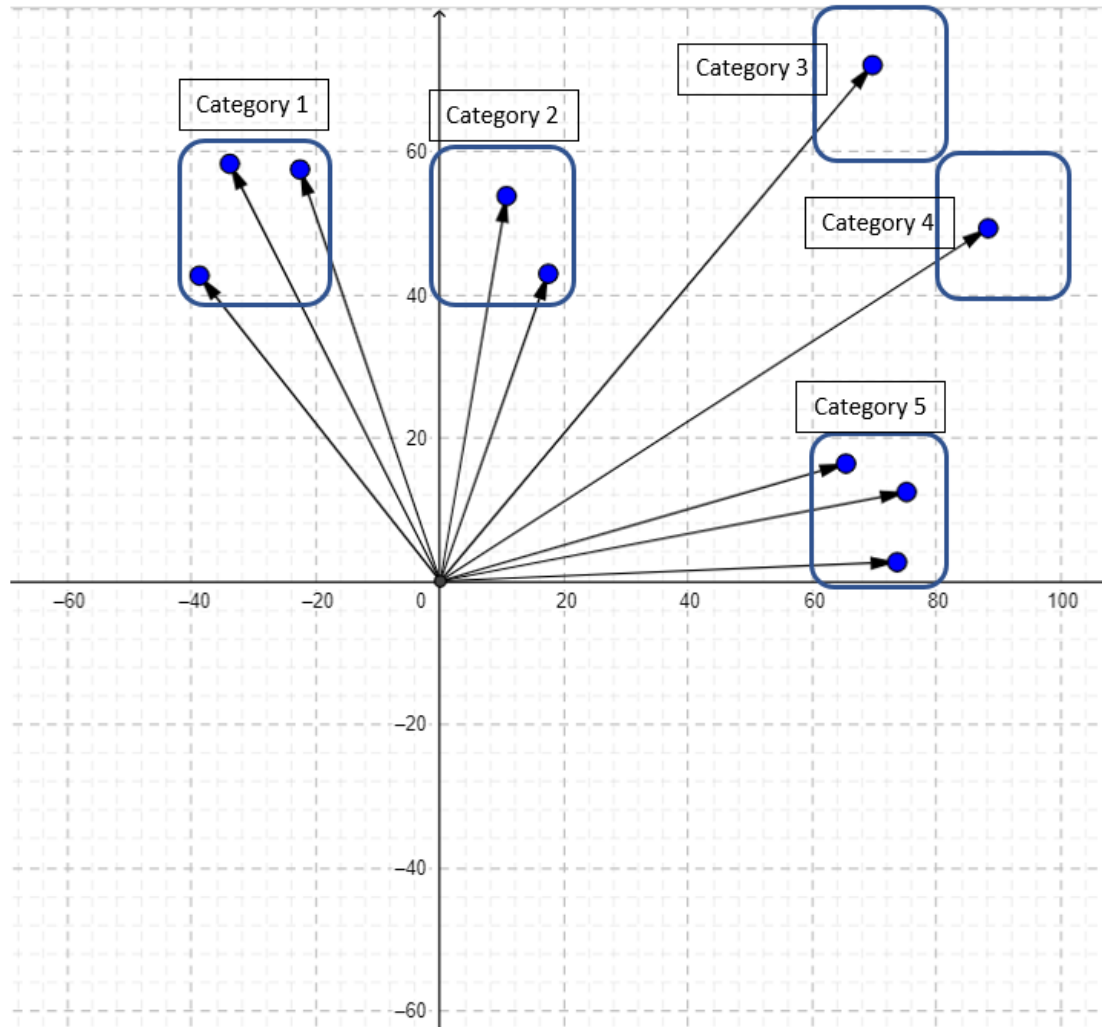


Figure 33: Grouping of Joint Vectors Based on 20x20 Grid Categories

So far, we created a dataset with 10 images, and scanned through the entire dataset for left shoulder-left elbow joint vectors and pasted 10 vectors onto 2D coordinate system and grouped them under 20x20 pixels grid. After grouping, 5 categories appeared. Number of vectors in each category is listed in Table 14.

Table 14: Number of Vectors in Each Category, Group

Category	Number of Vector
1	3
2	2
3	1
4	1
5	3

According to Table 14, in our algorithm, vectors in category 3 and category 4 will get the highest scores because they are one of a kind, most uncommon and unique vectors. However, vectors in categories 1 and 5 will get lowest scores because they are most common and easily found vector in the dataset. So far, we only performed pose complexity analysis for one joint (left shoulder to left elbow) but in real cases we perform for 9 different joint vectors in human body and multiply scores of 9 vectors for each image to get the final score. It can be thought as a 3D distribution plot.

Human objects with the highest scores will be most common and easy poses and will get least scores. However, people with least scores will be most unique and hard to find therefore, they will get highest pose complexity scores.

Numbers in Table 14 may seem trivial because it has only 10 images for demonstration purposes. However, when we apply this method datasets with thousands of images, distribution graph shows a complex behavior. For example, histogram given in Figure 34 shows pose complexity scores for a dataset with 30000 images with 4 people in each image. Histogram given in Figure 35 shows the individual pose complexity of the entire cropped person library with 49000 images.

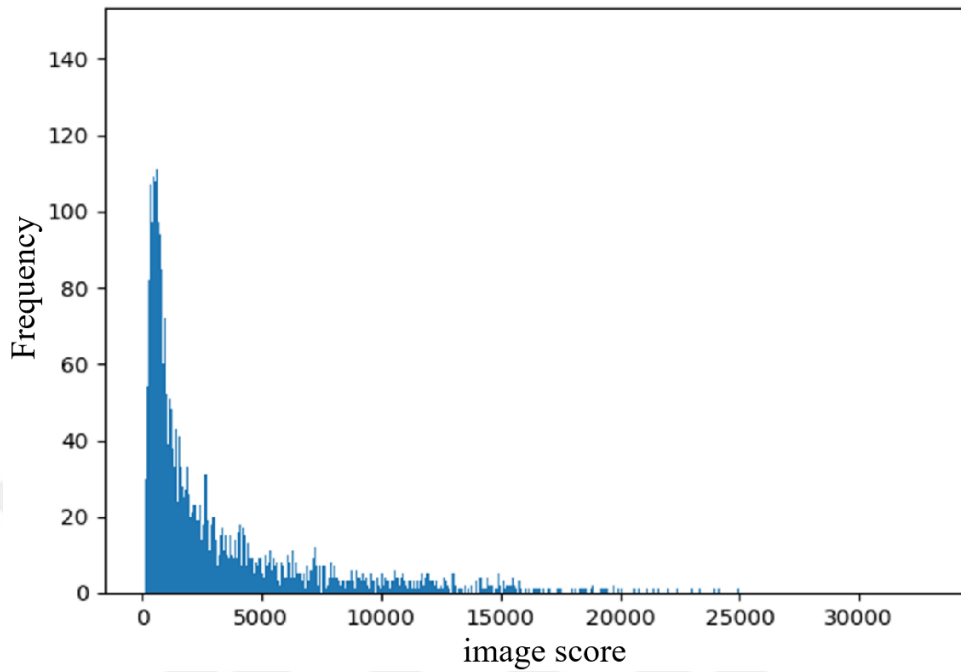


Figure 34: Histogram of Scores for 4 Person Case Using 30000 Images

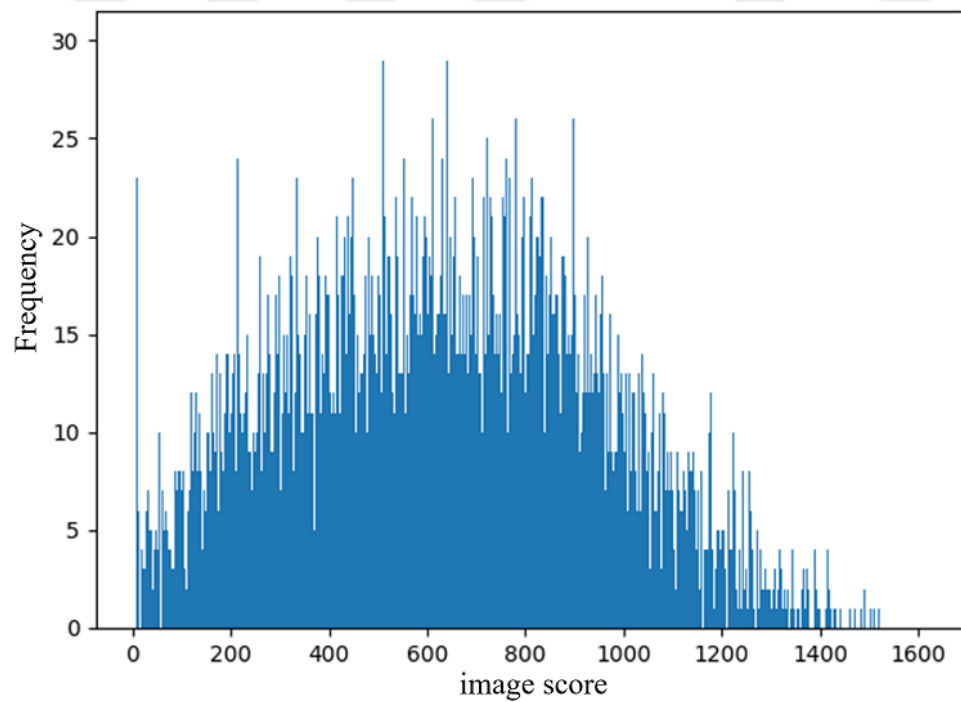


Figure 35: Histogram of Scores of Cropped Person Library (49000 Images)

Analysis on 30000 images with 4 people takes 10 seconds to run. In Figure 36 and Figure 37, 5 images with highest scores and 5 images with lowest scores are given after the analysis.



Figure 36: 5 Images with Highest Pose Complexity Scores

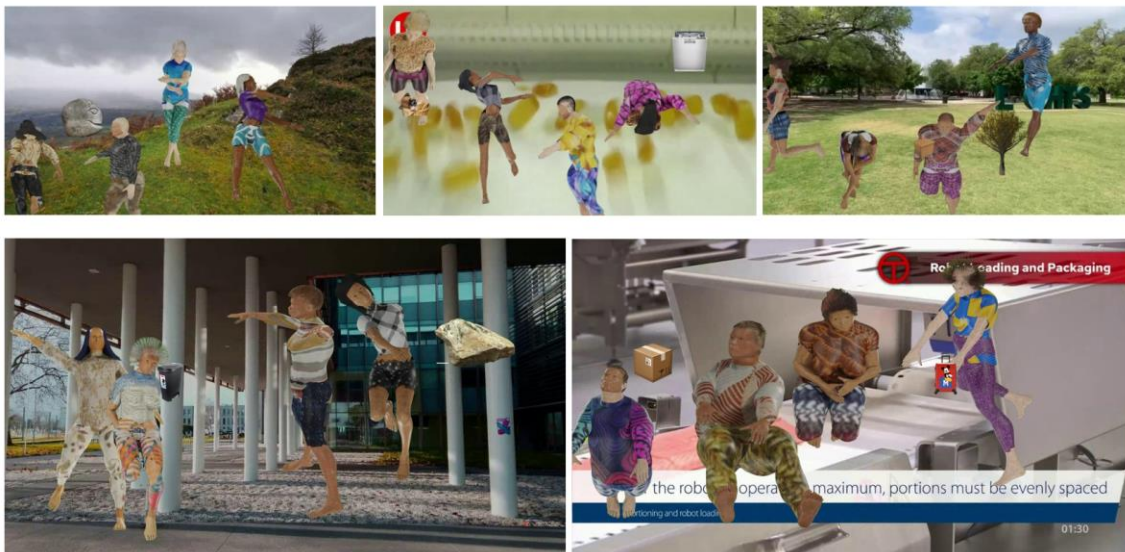


Figure 37: 5 Images with Lowest Pose Complexity Scores

As shown in Figure 36 and Figure 37 , images with most person to person interaction, most occlusion and truncation got the highest scores and images with least occlusion and common, easy poses usually taken straightly from front view got least scores.

Parameters effecting pose complexity scores are listed below:

- Number of images in each grid category
- Number of occluded joints in each person
- Number of truncated joints in each person

For each joint vector [i] in each human model [j], pose complexity analysis starts as below:

maxnum_i: 9x1 vector for each joint vector, it represents the number of vectors in the most crowded category for joint i (i: vector from left shoulder to left elbow etc.)

calnum_{ji}: 120000 x 9 vector for 120000 people and 9 joint vector. It represents number of vectors that fall into the same category, grid with vector coming from person j and joint vector i.

occluded_j: 120000 x 1 vector. Number of joints occluded or truncated for person j

personscore_j: 120000 x 1 vector. Score of each person

imagescore_k: 30000 x 1 vector. Final image scores of each image by summing person scores of 4 persons in each image.

$$\text{score}_j = \prod_1^i \left(\frac{2 * \text{maxnum}_i - \text{calnum}_{ji}}{\text{maxnum}_i} \right)^{1.2} \quad (1)$$

$$\text{personscore}_j = \text{score}_j * \left(\frac{9 + \text{occluded}_j}{9} \right)^3 \quad (2)$$

$$\text{imagescore}_k = \sum_1^4 (\text{personscore}_j) \quad (3)$$

score_j = initial score of person j without occlusion correction

personscore_j = Score of person j after making occlusion correction

imagescore_k = Score of image k by summing scores of 4 persons in each image.

3.3.5. Further Details About Three Dataset Generation Methods

Background Images

Street environment method places human models in 3D street. Therefore, no background image is required. However, they are required for plane video method and cropped person method. The background images change in every frame which means every image in a dataset contains different background image. Also, most of the images were taken from YouTube videos with various content such as indoor, outdoor, nature etc.

When creating background image library, special attention has been given to following items:

1. There can be no human in the background images because that disturbs training process. Only human in an image must be Blender human model.
2. To avoid people appearances in each video frame, we used a fast and highly accurate pose estimation model to detect people in different frames and discard that frame if there is a human in it. We provide the source code for this tool so that users can create their own background without human in it.
3. Videos contents are chosen from various domains and one frame was extracted in every 5 seconds of each video to avoid consecutive frame redundancy.

Animation and Human Movement in Blender

Blender software provides not only 3D design of objects but also animation and movement of objects in 3D space. To avoid overfitting during training, synthetic dataset must cover wide range of human body movements. Considering 178000 images in the dataset, pose and location of human models must be non-repeating. To be able to achieve that, animation feature of Blender was used.

To be able to move person or change pose of the human model in Blender, keyframes are used. Keyframes are pose and location instances of a human model. There are 4 different

parameters stored in an instance. These are:

- Full human body location information
- Full human body rotation information
- Location information of every joint
- Rotation information of every joint

By changing these 4 parameters, it is possible to locate and rotate human body anywhere in the whole image. It is also possible to change human pose completely by changing location and rotation of each joint. These variations create keyframes and each keyframe is used to create an animation of the moving human model. Two different keyframes are given in Figure 38.



Figure 38: Two Different Keyframes from Blender Animation

In Figure 38, human body poses for both left and right side was manually created and there are 22 frames between these keypoints. If start and end point of a person is provided, Blender can perform interpolation between start frame and final frame and fills 22 frame gap in such a way that it goes from frame 271 to frame 293 in a smooth way.

Since every start and finish keyframe is different, beginning and end of the interpolation is different so, movement between every keyframe is different. This method was used to ensure the nonrepeating human poses throughout dataset.

Dynamically Changing Parameters

So far, changing of background and variation of human pose in each frame have been discussed. To increase variability even further, some parameters inside Blender have been changed in each frame.

- In each frame, brightness of lights is changed randomly to create different lighting conditions.
- In each frame, t-shirt color and texture, pants color and texture, skin color and texture and hair color and texture of human model change.
- Images are taken by cameras from 5 different angles
- In each frame, scale of the model changes and that changes body ratio.

All these changes are adjusted via Python scripts which was provided in source code.

3.4. Testing Different Pose Estimation Models on Blender Dataset

In the duration of dataset generation, to quantitatively evaluate synthetic dataset, 4 different pose estimation models available on GitHub were used to test their performance on Blender Dataset. The purpose of this evaluation is to observe whether real world pose estimation models performs well on synthetic images created by Blender. To represent accuracy values, MPJPE metric [6] has been used. This is the L_2 norm over all the joints in all images.

$$\text{Error (MPJPE)} = \frac{(\sum_{k=0}^n ((x_{gt}-x)^2 + (y_{gt}-y)^2))^{0.5}}{n} \quad (4)$$

x: estimated x coordinate of joint

x_{gt} : ground truth x coordinate of joint

y: estimated y coordinate of joint

y_{gt} : ground truth y coordinate of joint

n: number of total joints in dataset

4 different models were used to perform this test. These models are:

Model 1: Lightweight Human Pose Estimation 3d Demo Pytorch [19]

Model 2: Posenet Pytorch [28]

Model 3: Deep High Resolution Net Pytorch [29]

Model 4: Pytorch Openpose [30]

These 4 models were tested on 200 Blender images and error values based on MPJPE metric are given in Table 15.

Table 15: Performance of 4 Different Models on Blender Dataset by mAP metric

Models	Error in pixels (MPJPE)	Detection Ratio (%)
Model 1	5.69	91.92
Model 2	3.03	95
Model 3	5.18	84.62
Model 4	4.35	92.08

In Table 15 above, error in pixel means MPJPE error over the entire Blender dataset. Detection ratio is the percentage of joints detected in whole dataset. For example, Model 1 detected 91.92% of all joints in Blender dataset and in detected joints, error is 5.69 pixels based on MPJPE error metric.

These values are close to values in literature. This means pose estimation models trained on real world data can perform well on Blender simulation data as well. Therefore, a

human pose estimation model trained on Blender dataset may work on real world data as well. This thesis was checked in later chapters.

3.5. Creation of Blender Dataset

So far, the way to build human model and 3D environment has been explained. After completing these steps, dataset was created.

In the dataset, there are 178000 images and a single .json annotation file. .json file contains joint locations in each image as x and y pixel coordinate location. Blender software has a 3D environment inside but created images needs to have 2D joint locations. Therefore, 3D environment was projected into 2D images to create dataset. This means 3D (x,y,z) coordinate of joints in Blender software, was converted into 2D (x,y) coordinate in images.

Camera internal parameters and projection matrix were used for this operation by assuming pinhole camera model. Visual representation of pinhole camera model is given in Figure 39.

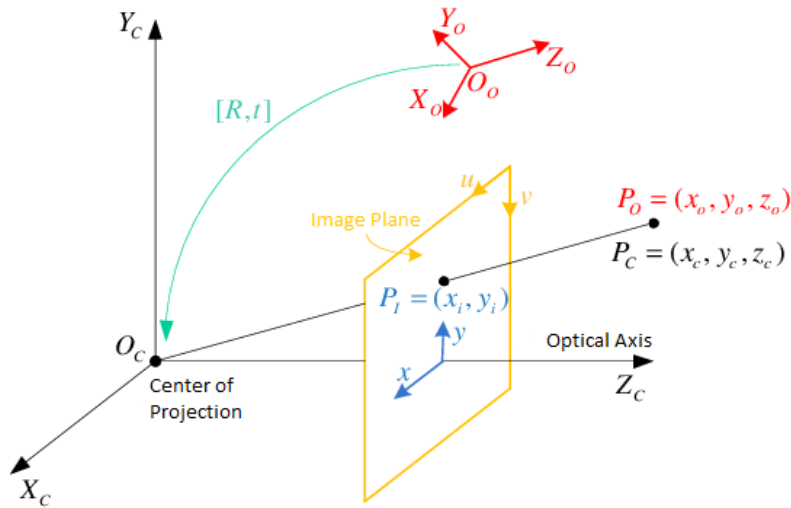


Figure 39: 3D to 2D Camera Projection Procedure [31]

P_o = original 3D coordinates of object point $[X_p, Y_p, Z_p]$

O_o = original 3D coordinates of camera $[X_o, Y_o, Z_o]$

P_c = coordinates of object point converted from P_o $[X_c, Y_c, Z_c]$

O_c = Center of projection $((0,0,0)$ coordinates in converted coordinate system)

In Blender, camera position and rotation information are available, and object point position information is also available. Focal length and image plane size is also available. To be able to convert 3D coordinates into 2D pixel coordinates following formula was used. Equation 5 provides a simplified version of camera calibration equation.

$$s * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r11 & r12 & r13 & -X_o \\ r21 & r22 & r23 & -Y_o \\ r31 & r32 & r33 & -Z_o \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \quad (5)$$

In Equation 5, $[X_p, Y_p, Z_p, 1]^T$ matrix is the 3D coordinates of object points. In this case, reference coordinate system is Blender original coordinate system. However, it needs to be converted into camera coordinate system and rotation and translation matrix $[R|T]$ was used for this operation. Rotation matrix is the 3x3 Euler rotation of camera with respect to Blender coordinate system. Translation matrix is a 3x1 matrix and it contains 3D coordinates of camera with respect to original Blender coordinate system. Rotation and translation matrix is augmented as shown in Equation 5.

Note that in Equation 5, object point coordinates are converted into camera coordinate system coordinates. In this case, reference coordinate system is taken as camera coordinate system but not Blender coordinate system. It is explained in Equation 6 below:

$$\begin{bmatrix} X_c \\ Y_c \\ Z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r11 & r12 & r13 & -X_o \\ r21 & r22 & r23 & -Y_o \\ r31 & r32 & r33 & -Z_o \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} X_p \\ Y_p \\ Z_p \\ 1 \end{bmatrix} \quad (6)$$

When object point position in camera coordinate system is found, the projection on image plane has to be found. Cameras in Blender software works with pinhole camera model. Therefore, this model is used to get 2D coordinates in image plane. Another visual

representation of pinhole camera model is given in Figure 40.

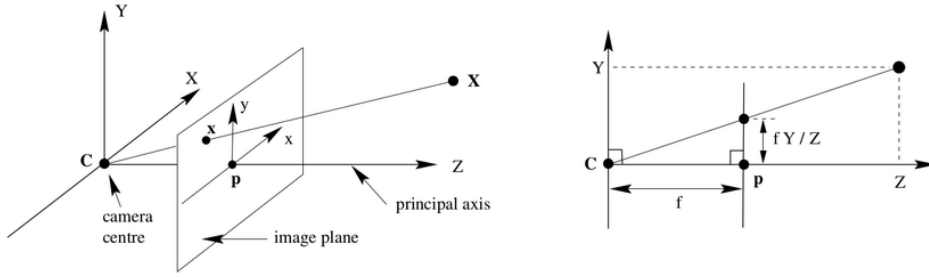


Figure 40: Representation of Pinhole Camera Models (HeadVision, 2019)

To be able to use pinhole camera models, internal camera matrix is used as shown in Equation 5. To perform similarity in triangle following equations are used:

$$s * \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} fx & 0 & cx \\ 0 & fy & cy \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Xc \\ Yc \\ Zc \\ 1 \end{bmatrix} \quad (7)$$

By performing matrix multiplications in Equation 7, u and v values were found and these are pixel coordinates of object points in created image. s value is a constant and it is found by the image plane size that can be adjustable in Blender software.

In this chapter, Blender features and methods to create synthetic dataset have been discussed. To create a rich dataset, different randomization methods have been applied. In the next chapter, details about usage of synthetic dataset on training and accuracy results have been discussed.

CHAPTER 4

4. TRAINING MODELS with BLENDER DATASET

In chapter 3, methods and procedures to create synthetic dataset with high variability have been discussed. By applying these procedures, synthetic dataset was created. This dataset consists of 178000 images and a single annotation file in .json format. However, to be able to utilize synthetic dataset in training, annotation file was converted to MPII dataset format. Details about files and training has been discussed in next parts.

4.1. Blender Annotation File Details

Blender annotation file is in csv format, and it contains 18 joint coordinates for every human model in the image. In the final dataset, .csv file is converted to .json file to be suitable with MPII dataset format. Name of the joints are given in the Table 16.

Table 16: Order and Name of Joints in Blender .csv Dataset

Joint Number	Joint Name	Joint Number	Joint Name
1	Top of the head	10	Right Wrist
2	Bottom of head	11	Chest
3	Left Thorax	12	Pelvis
4	Left Shoulder	13	Left Hip
5	Left Elbow	14	Left Knee
6	Left Wrist	15	Left Ankle
7	Right Throat	16	Right Hip
8	Right Shoulder	17	Right Knee
9	Right Elbow	18	Right Ankle

4.2. Training Models on Blender Dataset

Blender dataset is a custom-made dataset, and it can be converted into other dataset formats. Generally, different datasets in literature contains 16-17 joints in them and their joint order from first joint to last joint may differ. Also, file format and the way coordinates are presented may differ.

After literature survey about datasets, to evaluate our dataset's performance, two datasets from literature have been selected. These are MPII dataset and Crowdpose. Especially, MPII dataset is one of the most widely used human pose estimation dataset and there is more information and source code available on internet about this dataset compared to other datasets. MPII dataset contains over 25000 images of people performing different daily activities. Its annotation file formats are .mat format and .json format. Crowdpose dataset contains 20000 images from crowded scenes in daily activities. Its annotation file is a .json file as well.

Required information for MPII dataset is listed below:

- Joint visibility
- Joint locations
- Image file name
- Scale of human body in image with respect to 200 pixels
- Center coordinates of human body

All this information can be extracted from Blender software and Crowdpose dataset as well. Therefore, original Blender annotation file with .csv format were converted into MPII dataset format and Crowdpose dataset format was also converted to MPII format to be used in training later.

To be able train human pose estimation model, firstly, training model must be selected. Model can be trained from scratch or training can be resumed on a previously trained model. Then, training source code needs to be found. Source codes for models trained on MPII dataset are available on GitHub. Among them, 4 approach were selected. These are:

1. Epipolarpose [32]
2. Pytorch Pose [33]
3. Human Pose Estimation Pytorch [34]
4. HRNet [35]

After working on four different source code above, 3rd and 4th source codes were selected. It was possible to train first two pose estimation models, but they were excluded due to some limitations.

4.3. Details about Training

As explained in previous chapter, effectiveness of Metupose dataset has been tested via two pose estimation models and two pose estimation datasets. Models are Pose Resnet50 and HRNet. Datasets are MPII and Crowdpose dataset.

General procedure to measure effectiveness can be described as follows:

1. Firstly, Metupose dataset was combined with MPII dataset
2. ‘Original Pose ResNet50’ model has been trained further for 100 epochs with the combined dataset in step 1 by resuming training
3. Same ‘Original Pose ResNet50’ model has been trained further for 100 epochs with only MPII dataset
4. Performance of two trained models in step 2 and step 3 are tested on 3 different datasets. These are:
 - a. MPII dataset
 - b. Crowdpose dataset
 - c. LSPet dataset

The example given above were performed for (Pose ResNet50 model + MPII dataset) bundle. Same procedure was performed on (Pose ResNet50 model + Crowdpose dataset) bundle and (HRNet model + MPII dataset) bundle. Three different experiments are summarized in Table 17.

Table 17: 3 Experiments to measure effectiveness of Metupose dataset

Experiment	Used Pose Estimation Model	Used Literature Dataset
Experiment 1	Pose ResNet50	MPII
Experiment 2	Pose ResNet50	Crowdpose
Experiment 3	HRNet	MPII

The reason to perform 3 different experiments is to ensure the effectiveness of our dataset in different conditions. If we only performed Experiment 1 for example, we would only know effectiveness on Pose ResNet50 model and MPII dataset, but we would not know effectiveness on HRNet model and Crowdpose dataset. This is important for generalizability of effectiveness of our dataset.

Pose ResNet50 Pose Estimation Model Overview

Selected source code for Pose ResNet50 model is Human Pose Estimation Pytorch by Microsoft which is the official implementation of Simple Baselines for Human Pose Estimation and Tracking study.

Original MPII dataset annotation file is in .mat format but in source code, it was converted to .json format. Both json annotation file and pretrained models were available in their source code. Architecture and framework of Pose ResNet50 model is given in Figure 41 and Figure 42.

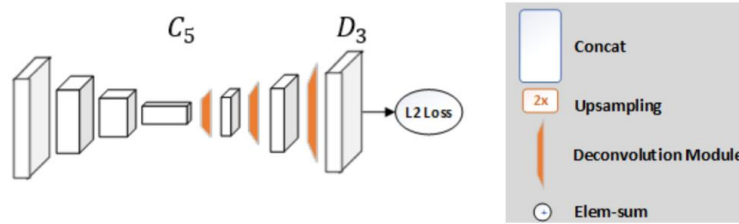


Figure 41: Architecture of Pose ResNet50 models used in Training [36]

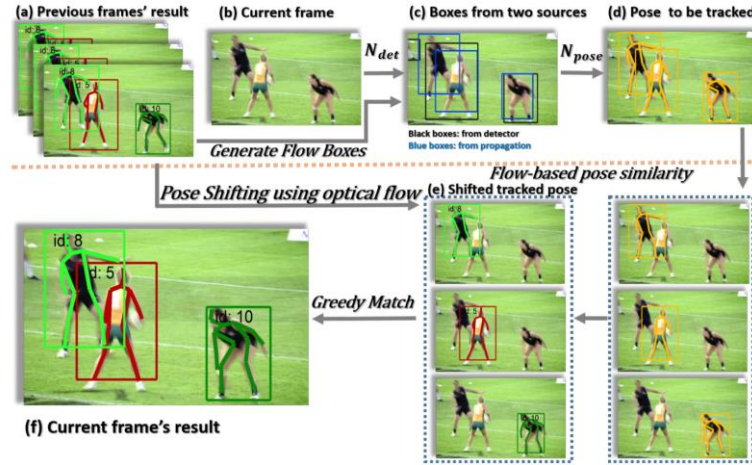


Figure 42: Framework of the Pose ResNet Model Used in Training [36]

In the training models, ResNet [37] was used as a backbone. Additional deconvolutional layers has been added to backbone and models were created.

Chosen source code has an accuracy metric called as PCKh. It is a different variation of PCK metric from a different study [38]. In this study, person bounding boxes were created and threshold bounding boxes are also created by multiplying person bounding box dimensions with a constant such as 0.1. For example, if person bounding boxes have dimensions such as 400 x 600 pixels, then, threshold bounding box dimension will be 40 x 60 pixels. Maximum of the two values which is 60 is taken. If the difference between estimated joint coordinate and ground truth joint coordinate is lower than threshold length (60 pixels) then, this estimation will be considered as correct. When this metric is applied to all images and all joints in a dataset, percentage of the correct labels will give PCK values. If 87% of all joints in a dataset was considered as correct, then pose estimation model will have PCK value of 87 in this dataset.

In source code, there was a slight change about accuracy metric. Instead of using full body to create threshold values, they used head segment length as reference length and 0.5 as the multiplying constant. This new metric is called PCKh (Percentage of Correct Keypoints-Head)

There were 6 different pretrained pose estimation models provided in the source code. In training and validation for Pose ResNet, only first model, pose_resnet50_256x256.pth.tar model was used. Accuracy values of six pretrained models from GitHub were tested on four different datasets and results are listed in Table 18.

Table 18: Accuracies of Pretrained Github Models on 4 Different Datasets

No	Model	Blender Acc (PCKh)	MPII Acc (PCKh)	LSPet Acc (PCKh)	Crowdpose Acc (PCKh)
1	Pose resnet50 256x256	39.5	88.5	26.1	59.1
2	Pose resnet50 384x384	39.5	89.1	24.2	61.0
3	Pose resnet101 256x256	40.2	89.1	25.8	59.6
4	Pose resnet101 384x384	40.7	90.0	24.2	61.6
5	Pose resnet152 256x256	41.3	89.6	24.2	60.5
6	Pose resnet152 384x384	43.5	90.2	25.4	63.6

HRNet Pose Estimation Model Preview

Selected source code for Pose HRNet model is the official implementation of Deep High Resolution Representation Learning for Human Pose Estimation.

Pretrained HRNet model was trained on MPII dataset and original MPII dataset annotation file is in .mat format but in HRNet source code, it was converted into .json format. Both json annotation file and pretrained models were available in their source code. Name of pretrained model is pose_hrnet_w32_256x256.pth and its architecture is given in Figure 43.

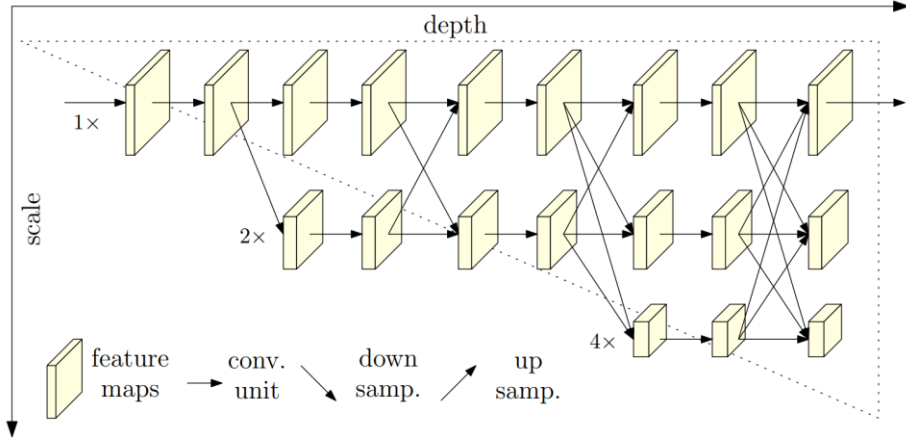


Figure 43: Architecture of pretrained HRNet Model

Both Pose ResNet50 and HRNet models in Github were pretrained on MPII dataset and their accuracy metric is PCKh.

Explanation for keywords used in accuracy results table is given in Table 19 and Table 20.

Table 19: Explanation for Different Datasets

No	Dataset Keyword	Dataset Meaning
1	MPII	Official MPII dataset containing 25000 people
2	Crowdpose	Official Crowdpose dataset, containing 26000 people from crowded scenes
3	LSPet	A portion of official LSPet dataset, containing 9000 people. Single person dataset with very difficult poses.
4	Blender	Sample from our Blender dataset containing 43200 people
5	BL_MP	Combination of BL ender and MP II dataset containing 68000 people
6	BL_CR	Combination of BL ender and CR owdpose dataset containing 69000 people

Table 20: Explanation for Different Training Conditions

No	Training Keyword	Training Meaning
1	FS 60E	Trained F rom S cratch for 60 epochs
2	RT_100E	R esumed T rainning on previously trained model for 100 epochs
3	ORG	Original model from Github

Sample model names based on combined keywords from the tables are given in Table 21.

Table 21: Definition of Sample Pose Estimation Model Names

No	Model	Model Meaning
1	PR--RT_100E--BL_MP	Pose ResNet model 100 epochs resume-trained on combination of Blender and MPII datasets
2	PR--FS_60E--BL_CR	Pose ResNet model trained from scratch for 60 epoch on combination of Blender and Crowdpose datasets
2	HR--RT_100E--BL_MP	HRNet model 100 epochs resume-trained on combination of Blender and MPII datasets
3	ORG--PR	Original pretrained Pose ResNet model from Github
4	ORG--HR	Original pretrained HRNet model from Github

Experiments and results are given in Table 22 to Table 24.

Experiment 1 (Pose ResNet model + MPII dataset) Bundle

Table 22: Accuracy Values for Experiment 1 (Pose ResNet50 + MPII Dataset)

No	Model	MPII (% PCKh)	Crowdpose (% PCKh)	LSPet (% PCKh)
1	PR--RT_100E--BL_MP	89.25	60.45	27.44
2	PR--RT_100E--MPII	88.73	58.76	25.83
3	ORG--PR	88.5	59.12	26.1

Experiment 2 (Pose ResNet + Crowdpose) Bundle

Table 23: Accuracy Values for Experiment 2 (Pose ResNet50 + Crowdpose Dataset)

No	Model	MPII (% PCKh)	Crowdpose (% PCKh)	LSPet (% PCKh)
1	PR--FS_60E--BL_CR	78.82	64.63	17.63
2	PR--FS_60E--Crowdpose	77.51	62.48	16.29

Experiment 3 (HRNet + MPII) Bundle

Table 24: Accuracy Values for Experiment 3 (HRNet + MPII Dataset)

No	Model	MPII (% PCKh)	Crowdpose (% PCKh)	LSPet (% PCKh)
1	HR--RT_100E--BL_MP	89.98	63.62	28.88
2	HR--RT_100E--MPII	89.91	62.03	27.70
3	ORG--HR	90.30	59.12	26.1

From the accuracies given in Table 22 to 23, findings are summarized below:

1. Regardless of the pose estimation model and dataset, our Blender dataset increased accuracies in all cases.
2. There was only a slight accuracy drop in Experiment 3 MPII test compared to original model. In that experiment original accuracy was already high (90.30 PCKh) and further training even in its own MPII dataset resulted in drop in accuracy. The reason could be original model could already be saturated and well-tuned for MPII dataset so, further training in MPII may disturb the model and result in very small decrease instead of an increase. Also, accuracy of our model is still higher than 100 epoch MPII resume training case.
3. Blender dataset increased accuracies both in training from scratch and resume training cases

4. In experiment 1, resume training in MPII dataset resulted in accuracy drop in Crowdpose and LSPet dataset but adding Blender dataset resulted in accuracy increase.

Experiment 4 (Changing Content of Metupose Dataset)

Apart from 3 experiment, another experiment has been performed to measure the effectiveness different Metupose dataset configurations. In this experiment, total number of people in the whole Metupose dataset has been kept same and image number in each image has been changed. In training source code, the important parameter is number of total people in all images but not the number of images itself. Results for (Pose ResNet50 and MPII dataset) bundle are given in Table 25.

Table 25: Performance of Metupose Dataset in Different Configurations

No	Model	MPII (% PCKh)	Crowdpose (% PCKh)	LSPet (% PCKh)
1	PR--RT_100E--BL_MP_1	89.07	59.71	27.19
2	PR--RT_100E--BL_MP_1234	89.09	60.48	26.88
3	PR--RT_100E--BL_MP_4	89.25	60.45	27.44
4	PR--RT_100E--MPII	88.73	58.76	25.83
5	ORG--PR	88.5	59.12	26.1

- PR--RT_100E--BL_MP_1 : Metupose dataset includes 1 person x 43200 images
- PR--RT_100E--BL_MP_1234 : Metupose dataset includes:
 - 1 person x 10800 images
 - 2 person x 5400 images
 - 3 person x 3600 images
 - 4 person x 2700 images
- PR--RT_100E--BL_MP_4 : Metupose dataset includes 4 person x 10800 images

From the accuracies given in Table 25, findings are summarized below:

1. Crowdpose is a multi person dataset and it contains crowded environment. LSPet

on the other hand is a single person dataset. When first and second Metupose dataset configuration was compared, first one (single person Metupose dataset) is better at LSPet and worse at Crowdpose.

2. Second configuration (mixed Metupose dataset) is better at Crowdpose and worse at LSPet dataset.
3. When first 2 configuration are compared as explained in steps above, results make sense because single person configuration performs better on single person dataset and mixed, multi-person configuration performs better crowded multi-person dataset.
4. However, instead of using mixed dataset, when we use only 4 people images, it performs better than any configuration in all cases. The reason for that could be increased occlusion and increase in complex interaction between different people.

Some may argue that adding new data to a dataset is expected to increase accuracy and therefore increase in accuracy due to additional Blender data is an expected outcome. This is correct. However, creating data in Blender is an automated process and it is completely free and does not take so much time. Furthermore, Blender provides all joint locations whether it is necessary or not for a specific dataset. This is especially useful because combining two different datasets from literature may be difficult or even impossible due to difference in the content of two datasets. Different dataset may include different number of joints, different location of joints labelled by different authorities.

Also, creating a new dataset for additional data manually is an expensive process but new and infinite amount data can easily be created via Blender. Because of these reasons, accuracy of a model can be increased almost effortlessly through Metupose dataset.

4.4. Deploying the Trained Model on Jetson Nano

Trained models are firstly tested on desktop computer through real time webcam video, later tested on Nvidia Jetson Nano.

As expected, desktop gave so much more fps compared to jetson nano and results are

given in Table 26 and Table 27.

Table 26: Real Time Fps Values of Our Trained Pose ResNet50 Model in Different Hardware

Hardware	Fps
Nvidia RTX3080	111.0 fps
Nvidia GTX1050ti	16.7 fps
AMD Ryzen9 5900x (CPU)	8.5 fps
Nvidia Jetson Nano (10W mode)	3.4 fps
Nvidia Jetson Nano (5W mode)	3.8 fps

Table 27: Real Time Fps Values of Our Trained Pose HRNet Model in Different Hardware

Hardware	Fps
Nvidia RTX3080	26.3
Nvidia GTX1050ti	13.1
AMD Ryzen9 5900x (CPU)	8.1
Nvidia Jetson Nano (10W mode)	2.8
Nvidia Jetson Nano (5W mode)	2.2

It was observed that Nvidia Jetson Nano was barely enough to track human. Also, during pose estimation, Jetson Nano gave over-current throttling error as shown in Figure 44. In some of the experiments, Nvidia Jetson Nano shut down itself because of high workload. Because these complex models were trained on Desktop computer it can be difficult for single board computer to run it non-stop. However, there is a way to increase fps by using TensorRT. This tool is specifically designed by Nvidia to convert Desktop trained models to Jetson compatible models. This tool provides an increase in fps between 1.2 times to 6 times the original fps. We could not achieve to convert our model via TensorRT but this can be done as future work.

Also, in desktop hardware, there was a small fps difference between ResNet50 and HRNet model in GTX1050ti and AMD CPU devices. However, there was almost 4 times the difference in RTX3080 hardware. Actual reason is not known but one of the possible reasons could be special deep learning technology used in latest RTX3080 GPU may work

completely optimized with ResNet50 model but not with HRNet model. Release dates for RTX3080 and GTX1050ti is 2020 and 2016 respectively. In the last a few years, Nvidia provided so much more optimization in deep learning applications with their new devices compared to 6-7 years ago. A sample image of Jetson Nano pose estimation studies is given in Figure 44.

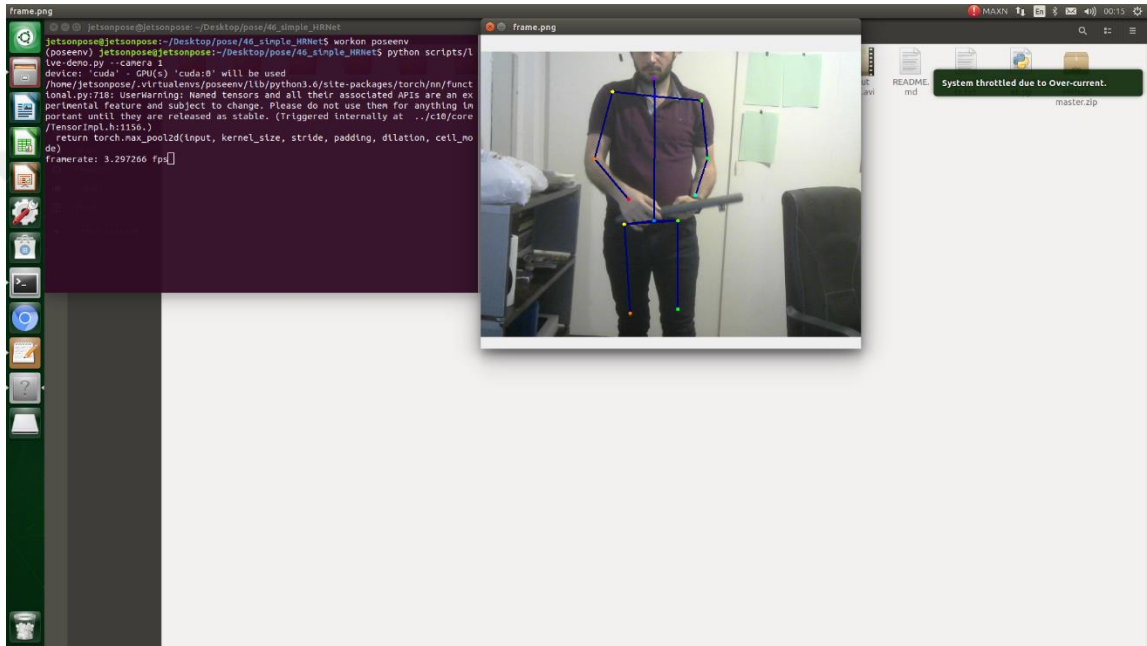


Figure 44: Throttle Error on Jetson Nano During Real Time Pose Estimation via ResNet50 Model



CHAPTER 5

5. CONCLUSION

5.1. General Conclusion

Human pose estimation is basically finding pixel location of joints in an image, and it is one of the most rapidly growing fields in machine vision. In this field, mostly, deep learning algorithms are used for maximum accuracy and flexibility. Although deep learning models give accurate results in most of the cases, there are still some cases where even some advanced models fail. These cases can be listed as: occlusion, multiple people in an image, interactions between people etc. To be able to overcome these problems pose estimation models need to be robust and flexible in different conditions and it needs to be trained on a wide variety of data to increase performance. However, creating a dataset from real world images is an expensive and time-consuming process and created data may be subject to errors because all joints are manually labelled by a human. To overcome these problems, a parametric synthetic dataset approach is proposed where existing datasets can also be enhanced with these synthetic images. When pose estimation models trained only on real world images were tested on simulation data created in a 3D graphics software, accuracies were satisfactory, and they were even close to the real-world accuracies. This means, once a model performs well on real world data, it does not differentiate simulation data from real world data and works on both. Therefore, our hypothesis is that when a model is trained only on simulation data, it can work on real world data as well since opposite case is observed to be true. To be able to test that, we created a synthetic pose estimation dataset in Blender 3D software and tested the effectiveness of our dataset with two different pose estimation model and 2 different datasets. During testing we trained different pose estimation models via Metupose dataset and observed changes in model accuracies. Our Metupose dataset led to accuracy increase

in all model/dataset cases. This means increasing a pose estimation model's accuracy by using Metupose dataset is possible and creating Metupose dataset is an automated and effortless process. We provide a complete 178000 image and 402000 person dataset and source code and necessary tools to create new dataset easily. By using Metupose dataset, we trained a pose estimation model and integrated the model into Nvidia Jetson Nano and Raspberry Pi camera system and results were satisfactory. This system can be used as a low-cost human tracking system in the future studies.

5.2. Future Work

Although Metupose dataset was successful for increasing accuracy of pose estimation models, there are some areas that can be improved in future studies.

Firstly, our trained model was run on Jetson Nano single board computer at 3.4 fps, this is barely enough to track a moving human object. To increase running fps, TensorRT module developed by Nvidia can be used.

MPII dataset contains total 25000 people and when we combine our Metupose dataset with MPII dataset, we added images with total 43200 persons out of total 402000 persons. For this case, future work could be increasing number of persons from Metupose dataset to observe if there is any increase in final accuracy.

All human models in our study were created by MbLab addon in Blender software and there are other tools and software to create even more realistic human models with different clothing style. Therefore, our whole study can be reproduced on another 3D design software platform to observe if there will be an increase in accuracy.

REFERENCES

- [1] Groos, D., Ramampiaro, H., & Ihlen, E. A. (2020). EfficientPose: Scalable single-person pose estimation. *Applied Intelligence*, 51(4), 2518–2533. <https://doi.org/10.1007/s10489-020-01918-7>
- [2] Artacho, B., & Savakis, A. (2020). UniPose: Unified Human Pose Estimation in Single Images and Videos. *2020 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr42600.2020.00706>
- [3] Fang, H. S., Xie, S., Tai, Y. W., & Lu, C. (2017). RMPE: Regional Multi-person Pose Estimation. *2017 IEEE International Conference on Computer Vision (ICCV)*. <https://doi.org/10.1109/iccv.2017.256>
- [4] Andriluka, M., Pishchulin, L., Gehler, P., & Schiele, B. (2014). 2D Human Pose Estimation: New Benchmark and State of the Art Analysis. *2014 IEEE Conference on Computer Vision and Pattern Recognition*. <https://doi.org/10.1109/cvpr.2014.471>
- [5] Lin, T. Y., Maire, M., Belongie, S., Hays, J., Perona, P., Ramanan, D., Dollár, P., & Zitnick, C. L. (2014). Microsoft COCO: Common Objects in Context. *Computer Vision – ECCV 2014*, 740–755. https://doi.org/10.1007/978-3-319-10602-1_48
- [6] Ionescu, C., Papava, D., Olaru, V., & Sminchisescu, C. (2014). Human3.6M: Large Scale Datasets and Predictive Methods for 3D Human Sensing in Natural Environments. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(7), 1325–1339. <https://doi.org/10.1109/tpami.2013.248>
- [7] Ganesh, P. (2019, May 25). *Human Pose Estimation : Simplified - Towards Data Science*. Medium. Retrieved January 30, 2022, from <https://towardsdatascience.com/human-pose-estimation-simplified-6cfd88542ab3>
- [8] Li, J., Wang, C., Zhu, H., Mao, Y., Fang, H. S., & Lu, C. (2019). CrowdPose: Efficient Crowded Scenes Pose Estimation and a New Benchmark. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr.2019.01112>

- [9] Cai, Y., & Tan, X. (2016). Weakly supervised human body detection under arbitrary poses. *2016 IEEE International Conference on Image Processing (ICIP)*. <https://doi.org/10.1109/icip.2016.7532427>
- [10] Brito, A. (2019, May 15). *Free download: Casual animated human scale* • Blender 3D Architect. <https://www.blender3darchitect.com/furniture-models/free-download-casual-animated-human-scale/>
- [11] Carica, A. (2021, October 12). *Unity 3d Person Model*. 3dmodelsz. Retrieved January 30, 2022, from <https://3dmodelsz.blogspot.com/2021/10/unity-3d-person-model.html>
- [12] Garreffa, A. (2021, March 17). *Unreal Engine MetaHuman tool makes super-realistic human models*. TweakTown. Retrieved January 30, 2022, from <https://www.tweaktown.com/news/77788/unreal-engine-metahuman-tool-makes-super-realistic-human-models/index.html>
- [13] *3D realistic rig - TurboSquid 1401132*. (2019, April 24). TurboSquid. Retrieved January 30, 2022, from <https://www.turbosquid.com/3d-models/3d-realistic-rig-1401132>
- [14] L. (2017, February 1). *People In Motion 2 Offers an Easy Way to Populate Scenes in C4D*. Lesterbanks. Retrieved January 30, 2022, from <https://lesterbanks.com/2017/02/people-in-motion-2-populate-c4d/>
- [15] *3D people character model - TurboSquid 1522543*. (2020, March 14). TurboSquid. Retrieved January 30, 2022, from <https://www.turbosquid.com/3d-models/3d-people-character-model-1522543>
- [16] Varol, G., Romero, J., Martin, X., Mahmood, N., Black, M. J., Laptev, I., & Schmid, C. (2017). Learning from Synthetic Humans. *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr.2017.492>
- [17] M. Fabbri, F. Lanzi, S. Calderara, A. Palazzi, R. Vezzani, and R. Cucchiara, "Learning to Detect and Track Visible and Occluded Body Joints in a Virtual World," *ECCV*, vol. abs/1803.08319, 2018
- [18] Patel, P., Huang, C. H. P., Tesch, J., Hoffmann, D. T., Tripathi, S., & Black, M. J. (2021). AGORA: Avatars in Geography Optimized for Regression Analysis. *2021 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr46437.2021.01326>

- [19] Osokin, D. (2021a, May 1). *GitHub - Daniil-Osokin/lightweight-human-pose-estimation-3d-demo.pytorch: Real-time 3D multi-person pose estimation demo in PyTorch. OpenVINO backend can be used for fast inference on CPU.* GitHub. Retrieved December 3, 2021, from <https://github.com/Daniil-Osokin/lightweight-human-pose-estimation-3d-demo.pytorch>
- [20] Tome, D. (2020, March 24). *GitHub - DenisTome/Lifting-from-the-Deep-release: Implementation of "Lifting from the Deep: Convolutional 3D Pose Estimation from a Single Image."* GitHub. Retrieved December 3, 2021, from <https://github.com/DenisTome/Lifting-from-the-Deep-release>
- [21] Zen, M. (2020a, July 14). *GitHub - MikeOfZen/Yet-Another-Openpose-Implementation: This project reimplements from scratch the OpenPose paper (Cao et al,2018), Using Tensorflow 2.1 and optional TPU powered training.* GitHub. Retrieved December 3, 2021, from <https://github.com/MikeOfZen/Yet-Another-Openpose-Implementation>
- [22] Uddin, M. (2020a, September 13). *GitHub - misbah4064/human-pose-estimation-opencv: Perform Human Pose Estimation in OpenCV Using OpenPose MobileNet.* GitHub. Retrieved December 3, 2021, from <https://github.com/misbah4064/human-pose-estimation-opencv>
- [23] Tobin, J., Fong, R., Ray, A., Schneider, J., Zaremba, W., & Abbeel, P. (2017). Domain randomization for transferring deep neural networks from simulation to the real world. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. <https://doi.org/10.1109/iro.2017.8202133>
- [24] Tremblay, J., Prakash, A., Acuna, D., Brophy, M., Jampani, V., Anil, C., To, T., Cameracci, E., Boochoon, S., & Birchfield, S. (2018). Training Deep Networks with Synthetic Data: Bridging the Reality Gap by Domain Randomization. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*. <https://doi.org/10.1109/cvprw.2018.00143>
- [25] Mehta, B., Diaz, M., Golemo, F., Pal, C.J., & Paull, L. (2019). Active Domain Randomization. *CoRL*.
- [26] *Free Blender Human Models.* (2022). TurboSquid. Retrieved January 30, 2022, from <https://www.turbosquid.com/Search/3D-Models/free/human/blend>
- [27] *About – MB-Lab Official Website.* (2021). Mblab. Retrieved May 20, 2022, from <https://mblab.dev/about/>

- [28] Wightman, R. (2019, August 14). *GitHub - rwightman/posenet-pytorch: A PyTorch port of Google TensorFlow.js PoseNet (Real-time Human Pose Estimation)*. GitHub. Retrieved December 3, 2021, from <https://github.com/rwightman/posenet-pytorch>
- [29] Xiao, B. (2021, May 19). *GitHub - leoxiaobin/deep-high-resolution-net.pytorch: The project is an official implementation of our CVPR2019 paper "Deep High-Resolution Representation Learning for Human Pose Estimation."* GitHub. Retrieved December 3, 2021, from <https://github.com/leoxiaobin/deep-high-resolution-net.pytorch>
- [30] Huang, Z. (2021, June 21). *GitHub - Hzzzone/pytorch-openpose: pytorch implementation of openpose including Hand and Body Pose Estimation*. GitHub. Retrieved December 3, 2021, from <https://github.com/Hzzzone/pytorch-openpose>
- [31] Dehkharghani, S. S., Pleshkova, G. S. (2014). Geometric Thermal Infrared Camera Calibration for Target Tracking by a Mobile Robot. *Comptes Rendus L'Academie Bulgare des Sciences*. 67. 109-114.
- [32] Kocabaş, M. (2019, July 26). *GitHub - mkocabas/EpipolarPose: Self-Supervised Learning of 3D Human Pose using Multi-view Geometry (CVPR2019)*. GitHub. Retrieved December 3, 2021, from <https://github.com/mkocabas/EpipolarPose>
- [33] Yang, W. (2021, February 10). *GitHub - bearpaw/pytorch-pose: A PyTorch toolkit for 2D Human Pose Estimation*. GitHub. Retrieved December 3, 2021, from <https://github.com/bearpaw/pytorch-pose>
- [34] Microsoft. (2019, April 10). *GitHub - microsoft/human-pose-estimation.pytorch: The project is an official implement of our ECCV2018 paper "Simple Baselines for Human Pose Estimation and Tracking(https://arxiv.org/abs/1804.06208)."* GitHub. Retrieved December 3, 2021, from <https://github.com/microsoft/human-pose-estimation.pytorch>
- [35] Sun, K., Xiao, B., Liu, D., & Wang, J. (2019). Deep High-Resolution Representation Learning for Human Pose Estimation. *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr.2019.00584>
- [36] Xiao, B., Wu, H., & Wei, Y. (2018). Simple Baselines for Human Pose Estimation and Tracking. *Computer Vision – ECCV 2018*, 472–487. https://doi.org/10.1007/978-3-030-01231-1_29

- [37] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. <https://doi.org/10.1109/cvpr.2016.90>
- [38] Yang, Y., & Ramanan, D. (2013). Articulated Human Detection with Flexible Mixtures of Parts. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(12), 2878–2890. <https://doi.org/10.1109/tpami.2012.261>
- [39] M. (2022). *GitHub - mustafa-ersoy/metupose-dataset: This code helps you create your own synthetic pose estimation dataset in MPII format using Python*. GitHub. Retrieved May 20, 2022, from <https://github.com/mustafa-ersoy/metupose-dataset>





CHAPTER 6

6. APPENDIX

All our source code is available in links below:

- <https://github.com/mustafa-ersoy/metupose-dataset>
- <https://drive.google.com/drive/u/2/folders/1-3NnpnKSBVgotMPqNe6fdZfMEEE7fPeE>
- <https://users.metu.edu.tr/kbugra/research/metupose/>

All links above contain same source code.