Portfolio

By: Jorge Victor Turriate Llallire

Summary

1. Introduction
2. Master’s thesis
3. Projects

INTRODUCTION

Profile

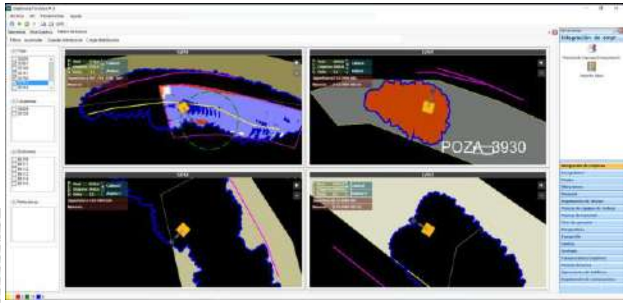
AI Researcher, Telecommunications Engineer and Web developer

Education

Bachelor in Telecommunications Engineering

Master of Science in Artificial Intelligence and Computer Vision

Dispatch



Provision

Minecare

Modular Mining Products

Professional Experience

Web developer

August 2019 – November 2019

Project Engineer Trainee

December 2019 – April 2021

Software, Deployment and Support Engineer

May 2021 – September 2023

FrontRunner
AHS

Project leadership

Troubleshooting

MASTER'S THESIS

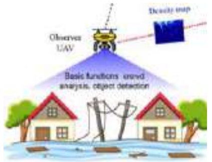
Curriculum Learning for Monocular Depth Estimation (MDE) in UAV Disaster Management

CONTEXT:

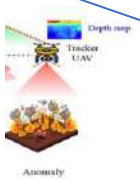
Disaster management:
Requires rapid and accurate decision-making

Unmanned Aerial Vehicles (UAV):
Collect real-time data, rapid aerial assessment and access hazardous locations

Monocular Depth Estimation (MDE)
Allows depth prediction from a single image



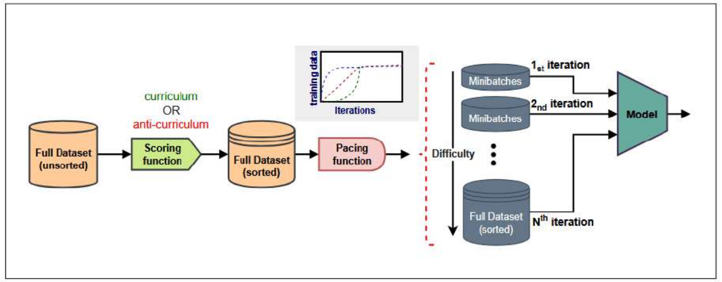
Crowd counting:
Identify regions of interest



Depth Estimation:
Help assess terrain and generate depth maps

MDE Problem:
Training MDE models is computationally intensive and often requires large, diverse datasets and a lot of training hours.

Curriculum Learning:
Offers a potential solution by accelerating convergence and improving generalization by mimicking human learning by presenting training data in a meaningful sequence

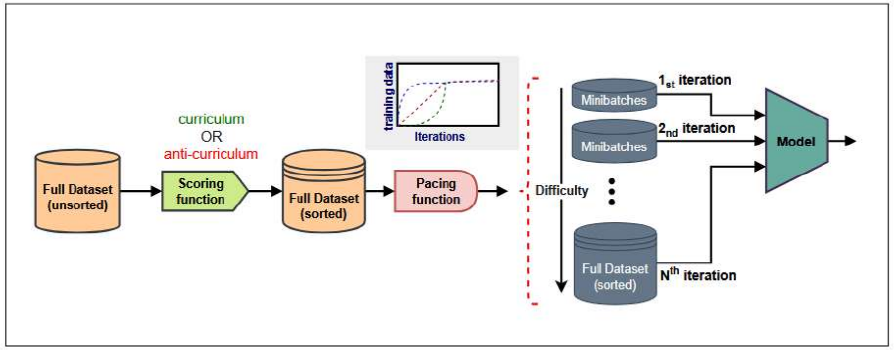


Will Curriculum Learning help the training of MDE models?

MISSION: Investigate the feasibility and impact of applying Curriculum Learning to Monocular Depth Estimation for UAVs used in disaster management scenarios.

OBJECTIVES:

- Review the current state-of-the-art MDE models suitable for UAVs.
- Adapt and structure training datasets to incorporate curriculum strategies.
- Compare training performance and accuracy between curriculum-based training and traditional training.



MONOCULAR DEPTH ESTIMATION (MDE)

Definition:
- Predict Depth map from a single RGB image
- Key Visual cues: Perspective, shading, texture, occlusion

One camera

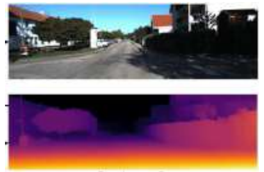
Input: 2D RGB Image

Output: 3D understanding (Depth map)

Comparison with other Depth Estimation models:
- Stereo Vision
- RGB-D
- Active sensors (LIDAR)

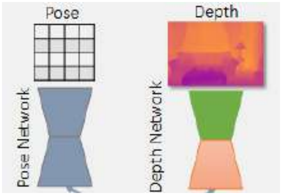
MDE has a **low cost**

Deep Learning paradigms for MDE:
- Supervised Learning



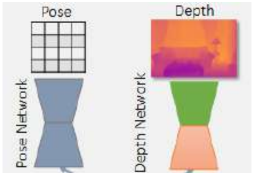
Training using the GT Depth map

- Unsupervised Learning (Self-supervised Learning)



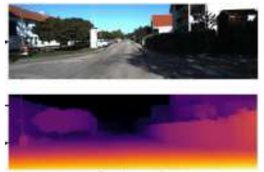
Training without any GT

- Hybrid approach



Pretrained network (SSL)

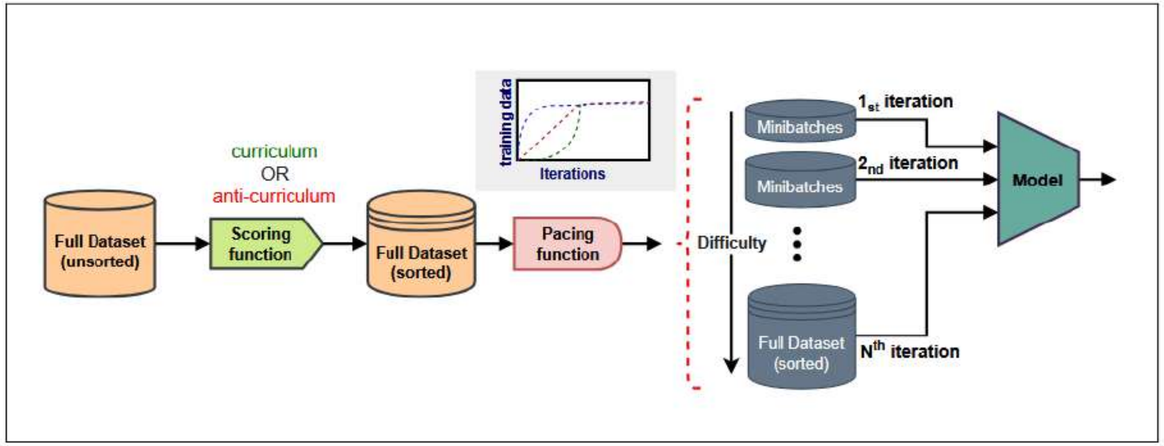
+



Fine-tuning (Supervised learning)

CURRICULUM LEARNING (CL)

Definition:
Training paradigm inspired by natural learning in humans.
Core principle: Structure the data in progressive manner based on the order.



Algorithm 1: Curriculum learning

Input: T steps, N samples, Initial weights w^0, training set $X = \{x_1, x_2, \dots, x_N\}$, pacing function $g: [T] \to [N]$, scoring function $s: [N] \to \mathbb{R}$, order $o \in$ {"ascending", "descending"}

$(x_1, x_2, \dots, x_N) \leftarrow sort\ X\ using\ s(x)\ in\ order\ o$

for $t = 1,2,3, \dots, T$ **do**

| $size \leftarrow g(t)$

| $X_t = X[1{:}size]$

| $w^{(t)} \leftarrow train\_one\_epoch(w^{(t-1)}, X_t)$ sampling uniformly mini batches

end for

Elements:

- **Scoring function**

Defines **the score** for a given training sample

The **difficulty** is defined with the comparison of scores

$$s(x_j, y_j) > s(x_i, y_i)$$

The loss is used to define the scores

$$s(x_i, y_i) = loss(f_w(x_i), y_i)$$

- **Order:**

✓ Curriculum: From lowest to highest score

✓ Anti-Curriculum: From highest to lowest score

Types of scoring function:

✓ Self-taught scoring function

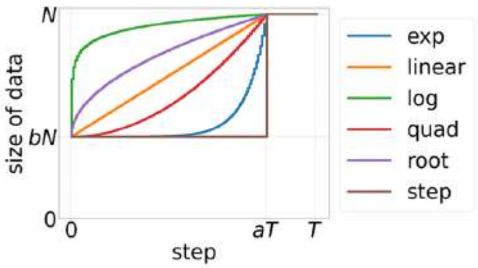
✓ Transfer scoring function

- **Pacing function**

Determines **the size** of the training data subset to be used at iteration T

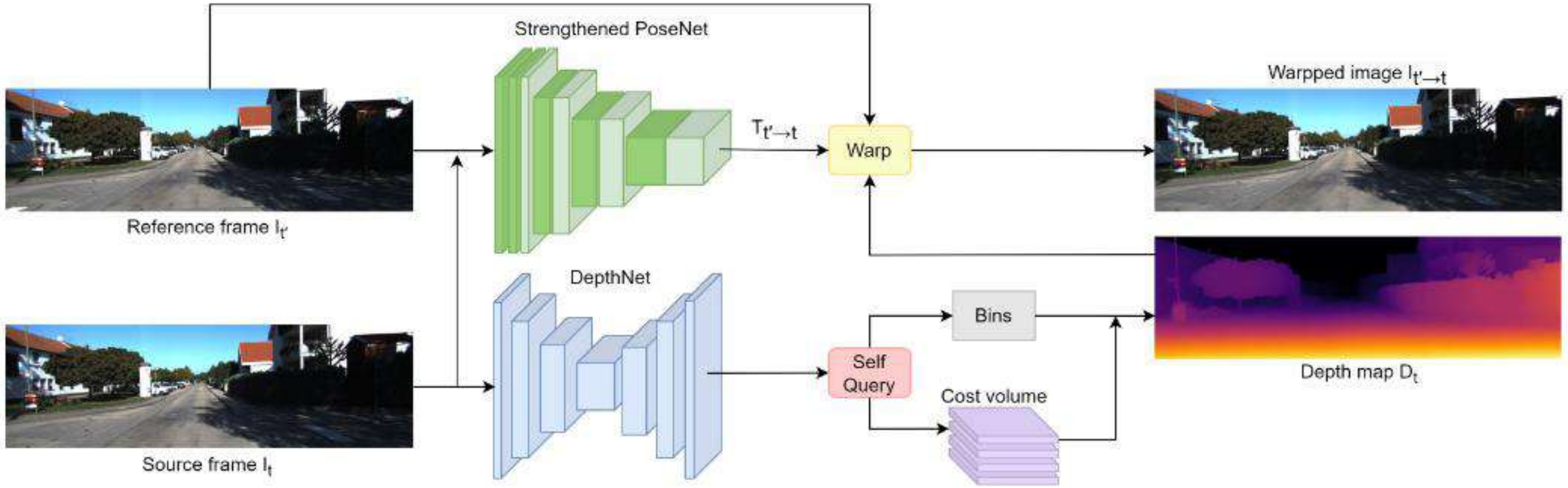
Name	Expression $g_{(a,b)}(t)$
linear	$Nb + N\frac{(1-b)}{(aT)}t$
quadratic	$Nb + N\frac{(1-b)}{(aT)^2}t^2$
root	$Nb + N\frac{(1-b)}{(aT)^{1/2}}t^{1/2}$
logarithmic	$Nb + N(1-b)(1 + 0.1\log\left(\frac{t}{aT} + e^{-10}\right))$
exponential	$Nb + \frac{N(1-b)}{e^{10}-1}(\exp\left(\frac{10t}{aT}\right) - 1)$
step	$Nb + N\lfloor\frac{x}{aT}\rfloor$

- a: Defines the speed in which the pacing function reaches the total number of samples
- b: fraction of training data used at the beginning



SPIdepth network

SOTA MDE network. It uses **self-supervised learning** to train its subnetworks.



Sub Networks:

- **Depth Net:** Infers depth maps from single RGB images by using an encoder-decoder framework.
- **Pose Net:** Predicts the relative pose (T') between the image I and the reference image I'

Loss function:

$$L = \mu L_p + \lambda L_s$$

- **Lp:** Masked photometric loss
- **Ls:** Per-pixel smooth loss

Training strategy:

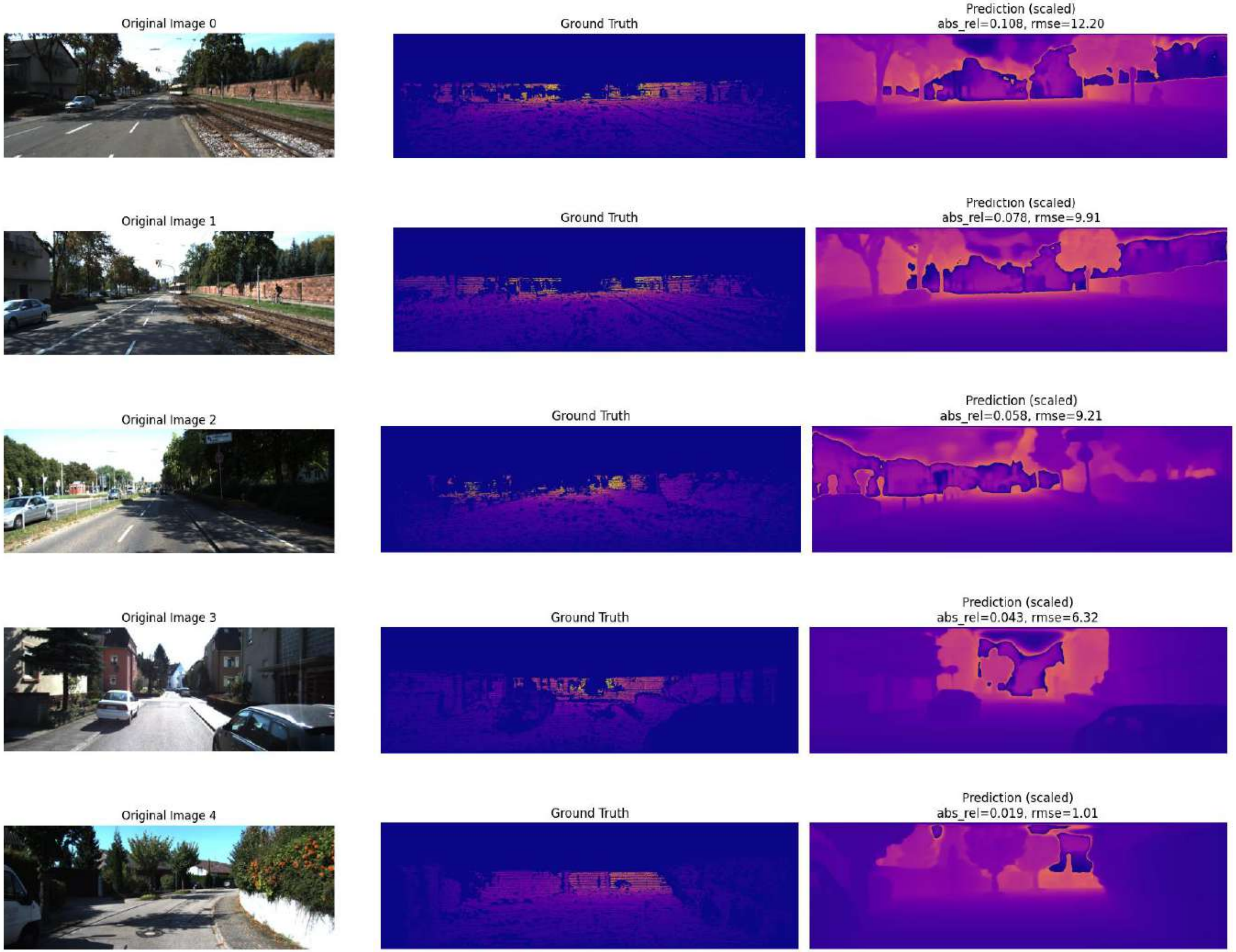
It uses a **hybrid learning** approach

- **Self-supervised training** on 75,000 images from KITTI
- **Supervised fine-tuning** on additional 25,000 images from KITTI

Inference on KITTI Eigen Split:

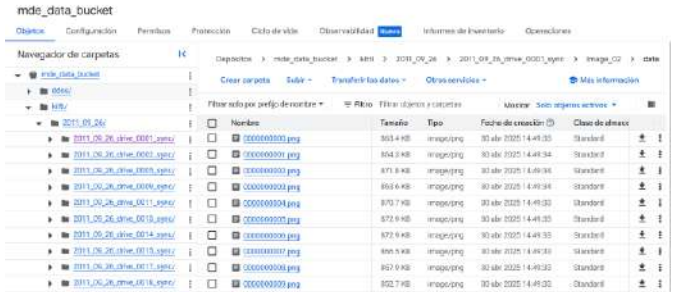
Absolute relative	Square relative	RMSE	RMSE_log	a1	a2	a3
0.066	0.416	3.359	0.147	0.945	0.974	0.986

Depth Map predictions – SPIdepth network pre-trained model



Dataset pre-procesing

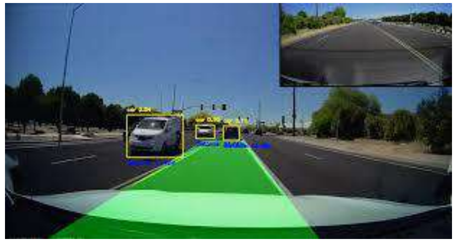
KITTI Data-set reduction



75,0000 images

Train split
(70,000 images)

Validation split
(5,000 images)



Detect car images using YOLOv8

Filter the images that contains 1-3 cars

New split files

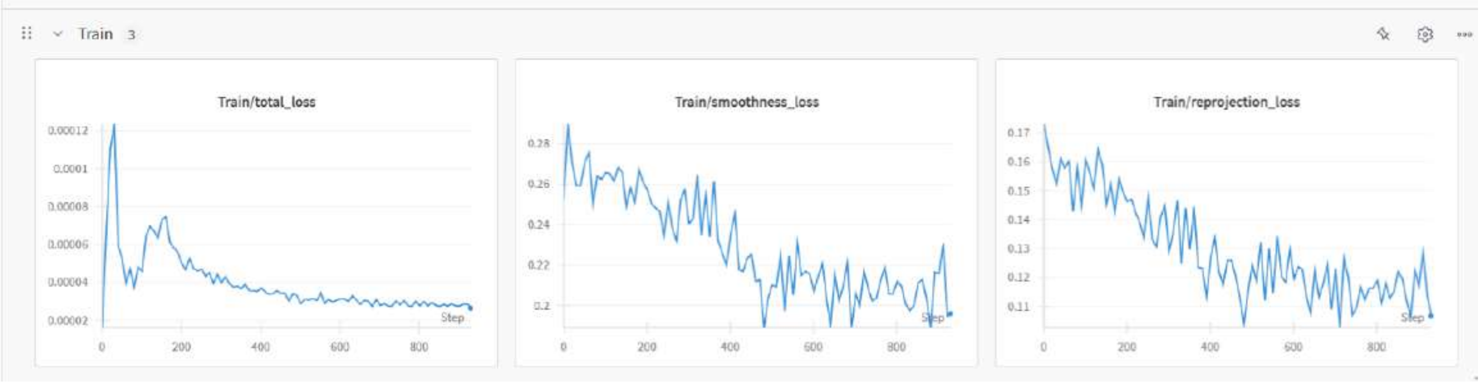
Train split file (2,500 images)

Validation split file (200 images)

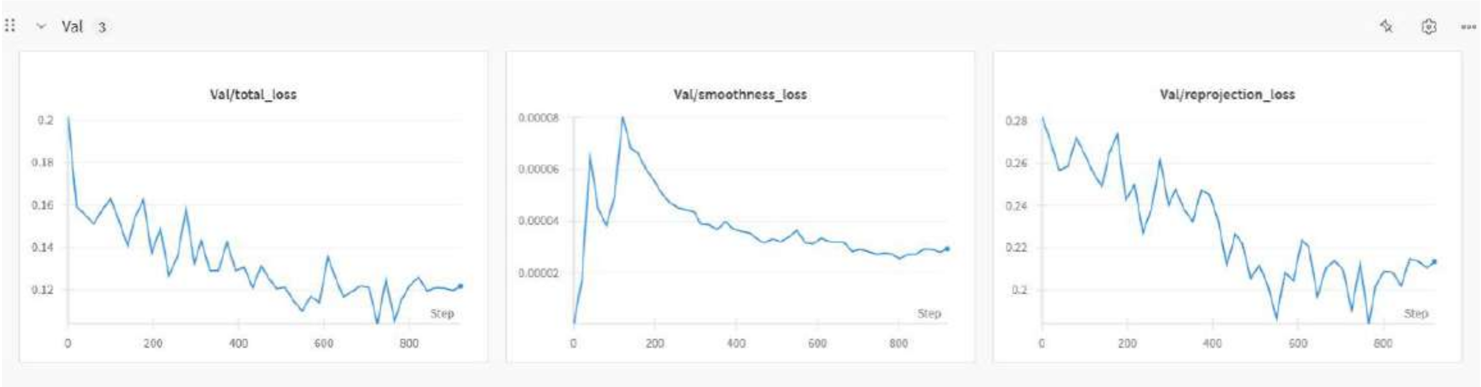
SPIdepth network – KITTI Baseline training

Hyperparameters

- Dataset: 2,500 train images
- Epochs: 6
- Batch size: 16
- Image resolution: 640x192
- Encoder backbone: ResNet-18 Lite



Converged around **0.000027** after **938 steps**



Converged around **0.12** after **938 steps**

Inference on Reduced validation set

Validation set of 200 images, filtered using YOLO

Absolute relative	Square relative	RMSE	RMSE_log	a1	a2	a3
0.157	1.142	5.481	0.223	0.795	0.933	0.976

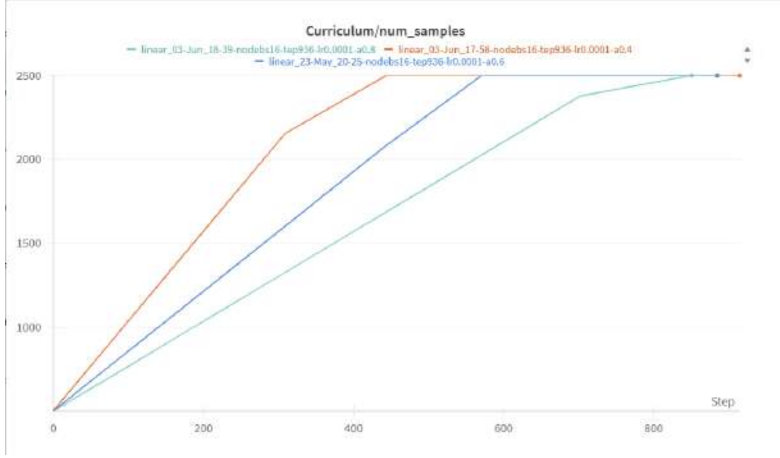
SPIdepth network – KITTI Curriculum Learning

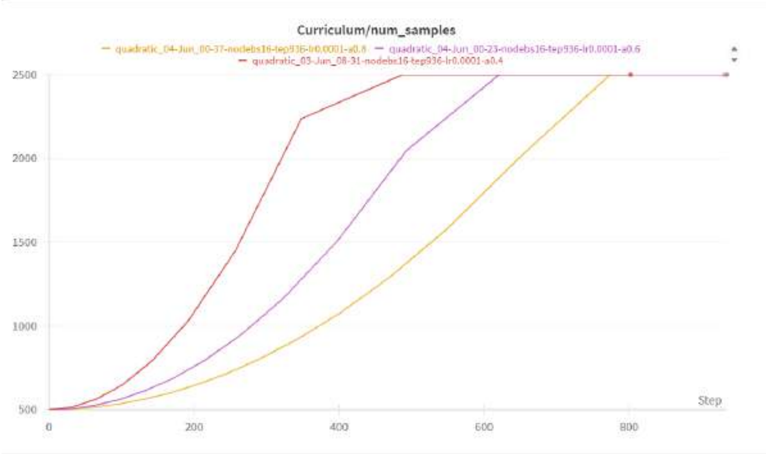
Scoring function $L = \mu L_p + \lambda L_s$

- Self-taught scoring function: Using saved weights from Baseline training (after epoch 3)
- Transfer scoring function: Pretrained SPIdepth model provided by the author

Pacing function

- T: 936 steps (6 epochs)
- N: 2500 samples
- b: 0.2 (fixed)
- a: {0.4, 0.6, 0.8} -> a controls the speed at which the full dataset is revealed during training





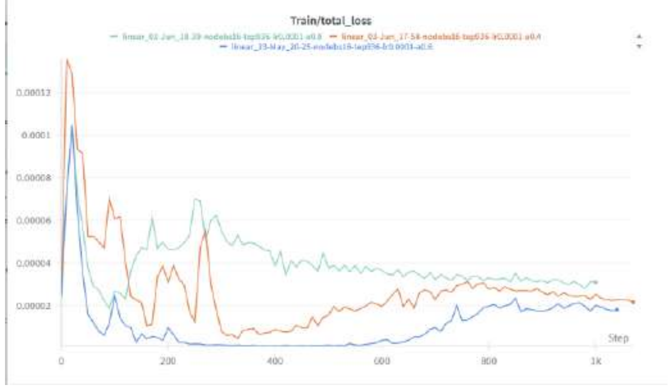
TRAINING EXPERIMENTS

10 experiments:

- **Pacing function hyperparameters (6 experiments):** Linear and Quadratic pacing function with different a values {0.4, 0.6, 0.8} and 936 steps
- **Faster convergence (3 experiments):** Linear pacing function with different a values {0.4, 0.6, 0.8} with less steps (468 steps).
- **Self-taught vs Transfer scoring (1 experiment):** Using a Linear pacing function with a=0.6

a	Absolute relative	Square relative	RMSE	RMSE_log	a1	a2	a3
0.4	0.180	1.333	6.606	0.257	0.726	0.906	0.967
0.6	0.242	2.775	10.313	0.415	0.581	0.793	0.884
0.8	0.156	1.113	5.339	0.220	0.797	0.936	0.978

Inference with **Linear Pacing function with a=0.8,** achieved a better result compared with Baseline training

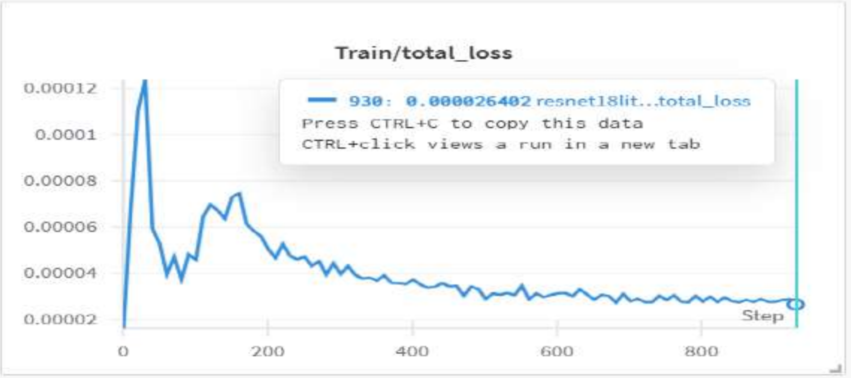


Training converged around **0.000017 (a=0.6)**

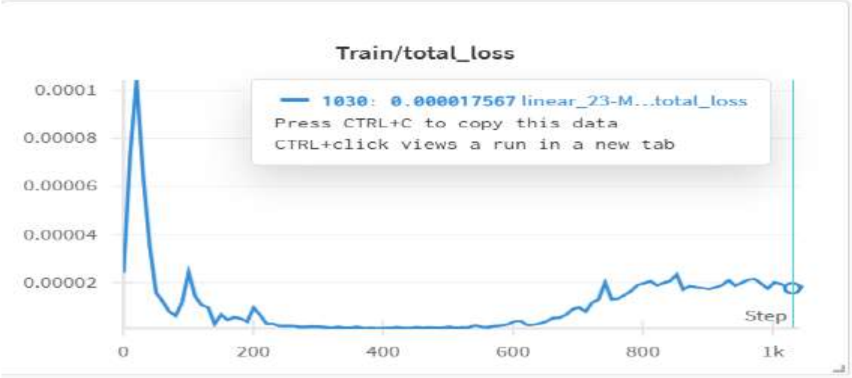
SPIdepth network – Curriculum Learning

CONVERGENCE IN TRAINING

Linear pacing function (a=0.6) reduced the Training loss from 0.000027 to 0.000017. (Better convergence)



Baseline training



Curriculum Learning

GENERALIZATION

Linear pacing function (a=0.8) generalized better for unseen data.

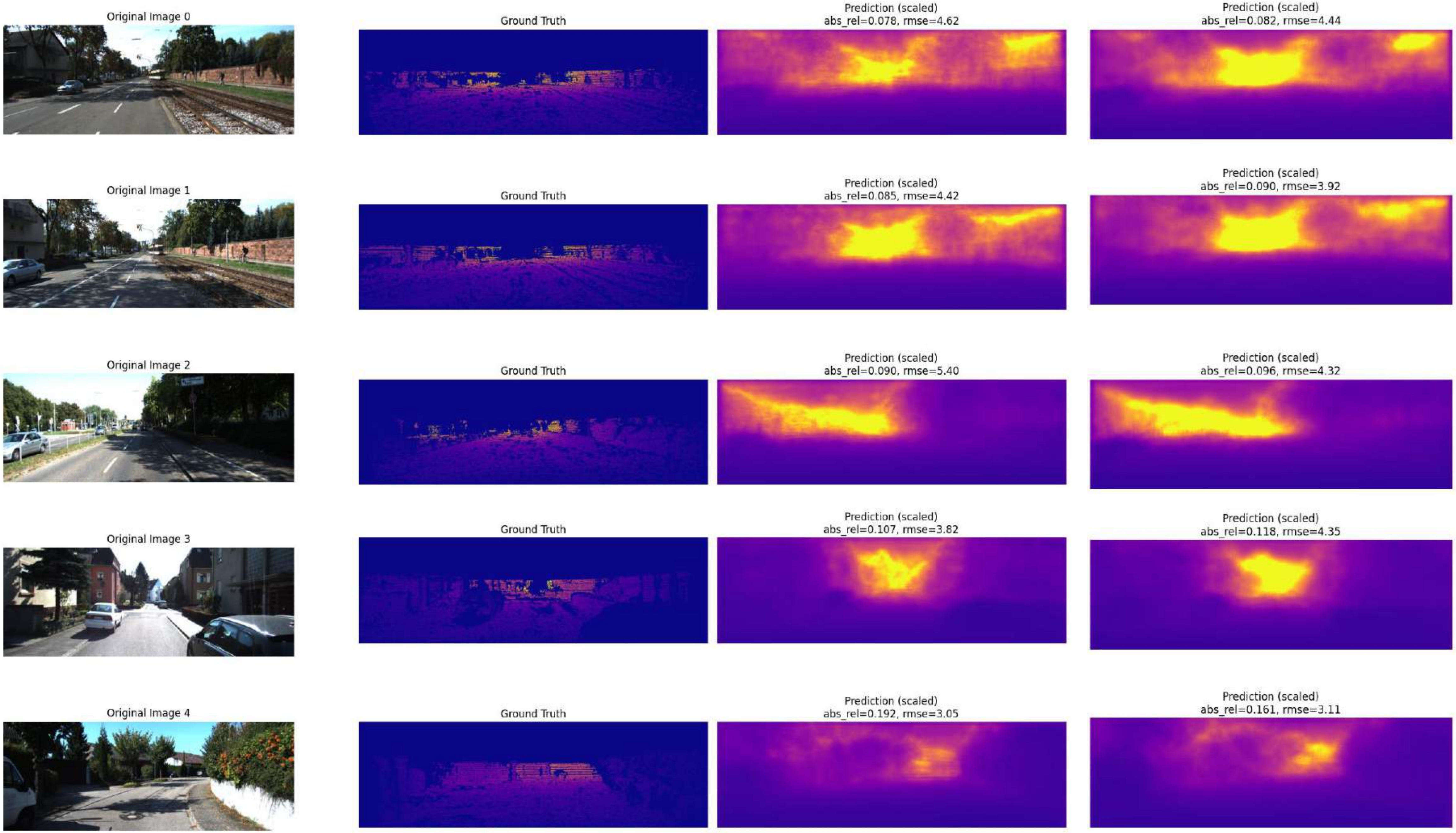
Absolute relative	Square relative	RMSE	RMSE_log	a1	a2	a3
0.157	1.142	5.481	0.223	0.795	0.933	0.976

Baseline training

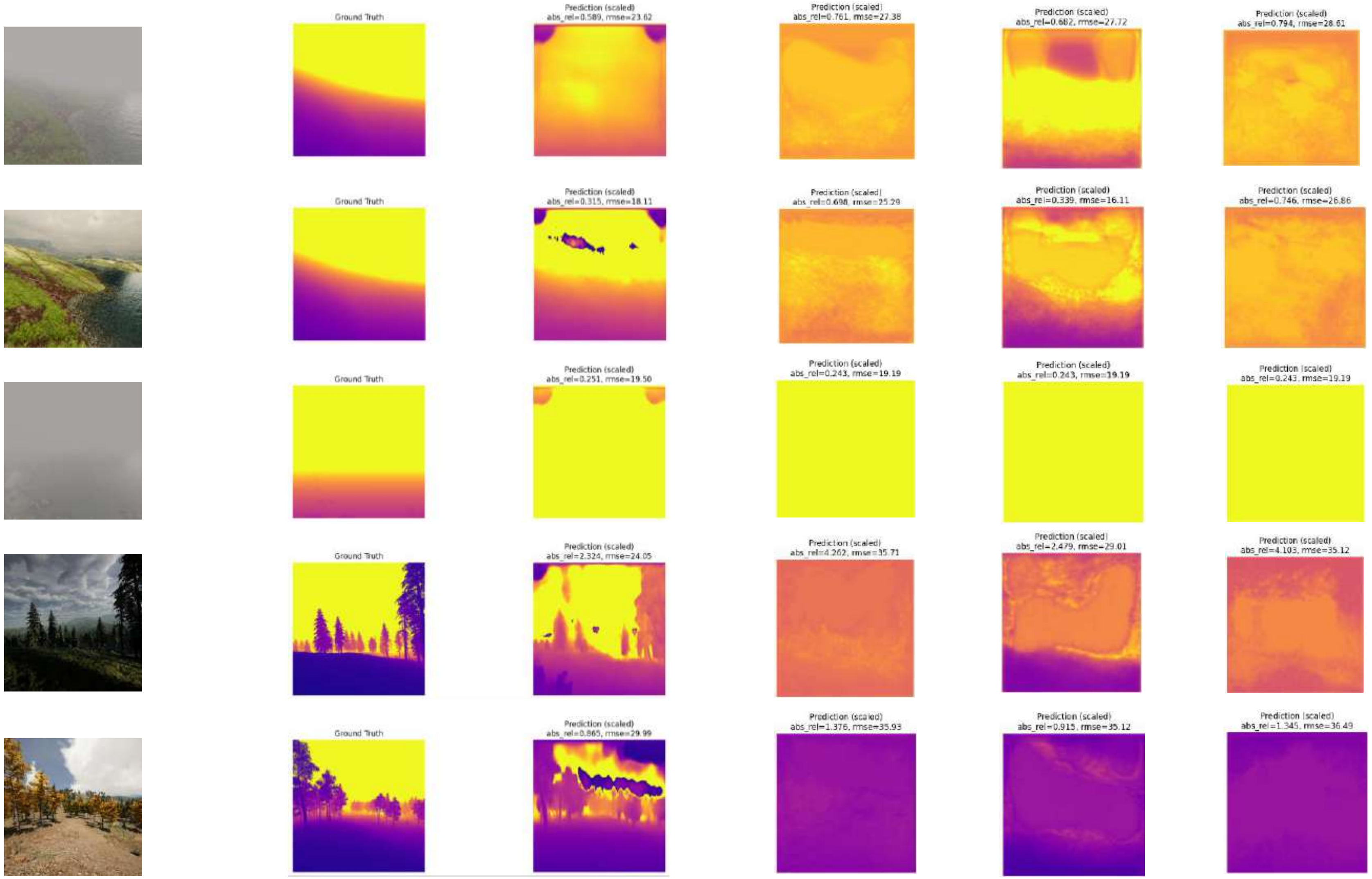
a	Absolute relative	Square relative	RMSE	RMSE_log	a1	a2	a3
0.4	0.180	1.333	6.606	0.257	0.726	0.906	0.967
0.6	0.242	2.775	10.313	0.415	0.581	0.793	0.884
0.8	0.156	1.113	5.339	0.220	0.797	0.936	0.978

Curriculum Learning

Depth Map predictions – KITTI Curriculum Learning Training (a=0.8 best generalization) vs Baseline Training



Depth map predictions MidAIR – SOTA SPIdepth vs Baseline Training ep 9 vs MidAIR Curriculum Learning Training (a=0.8 best generalization) ep 8 vs ep 9



Model	Dataset	Training	Epochs	Baseline Abs Rel	Baseline RMSE	CL Abs Rel	CL RMSE
SPIdepth	KITTI	Unsupervised	6	0.157	5.481	**0.156**	**5.339**
	MidAir	Unsupervised	9	1.126	**33.273**	**1.125**	33.371
Monodepthv2	KITTI	Unsupervised	–	–	–	–	–
	MidAir	Unsupervised	9	1.162	34.702	**1.157**	**33.449**

CONCLUSIONS

- The experiments demonstrated that Curriculum Learning not only led to faster convergence but also improved generalization in early stages of training. This behavior suggests that CL can accelerate convergence during the initial iterations, which is particularly advantageous in time-constrained scenarios such as disaster response, where rapid deployment is critical.

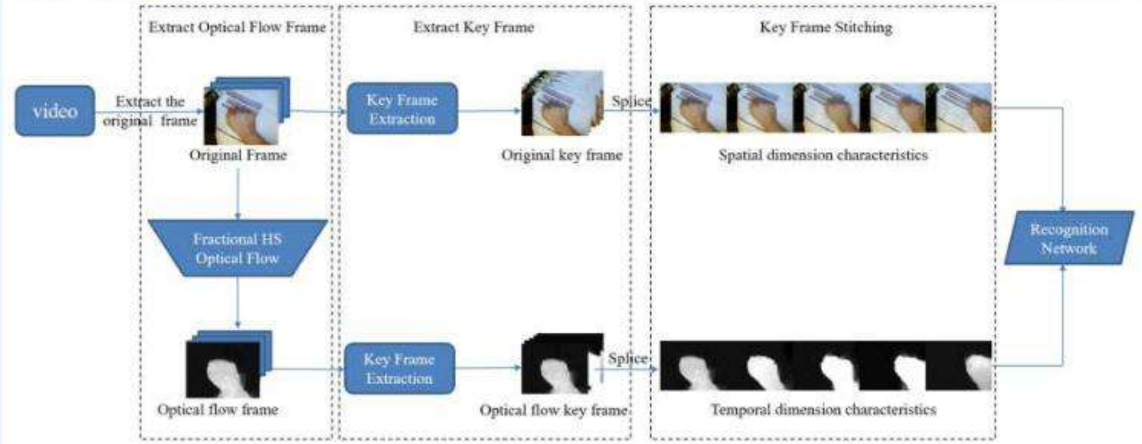
What is next?:

- Perform more experiments using Supervised Learning
- Publish these results in a journal paper

Projects

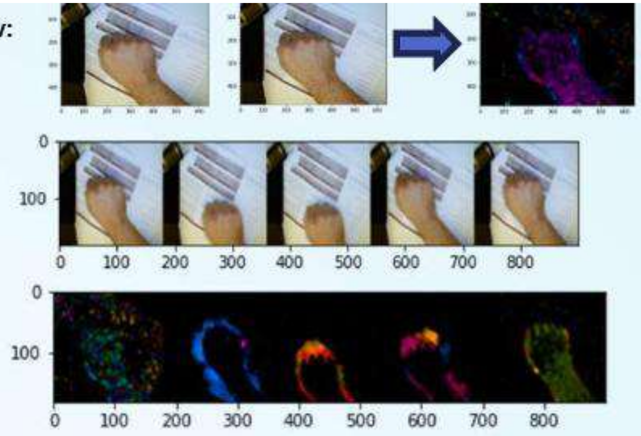
Real-time Hand Gesture Recognition for Interactive Video Conferencing

1. Machine Vision: Developing and training the CNN using Optical Flow and Key Frame Extraction techniques.

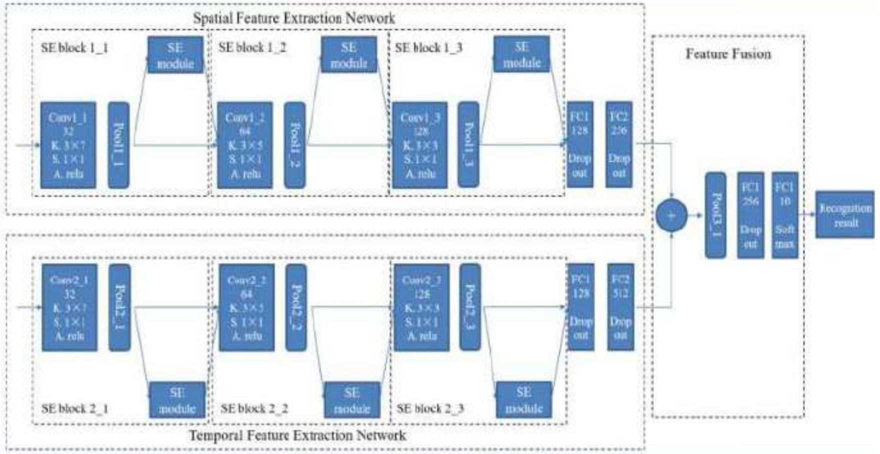


Optical Flow: Gunnar-Farneback flow

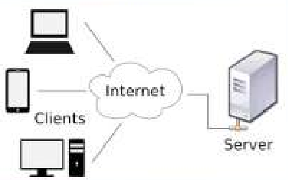
Key Frame Extraction: 5 frames Clustering

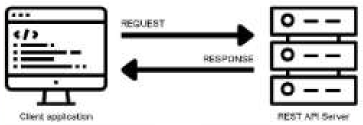


CNN architecture:



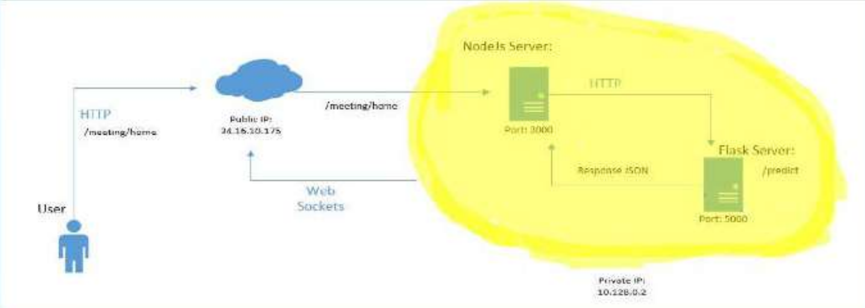
2. Web server development: Backend Server API server (Flask) and Frontend server NodeJS





Flask

3. Deployment: Hosting the application on Google Cloud Platform to ensure scalability and reliability.



Flask Server: Serves the API (/predict) endpoint at Port 5000

NodeJs Server: Serves the Web Server at port 3000

Projects

3D Reconstruction using Structure from Motion and NeRFs

. . .

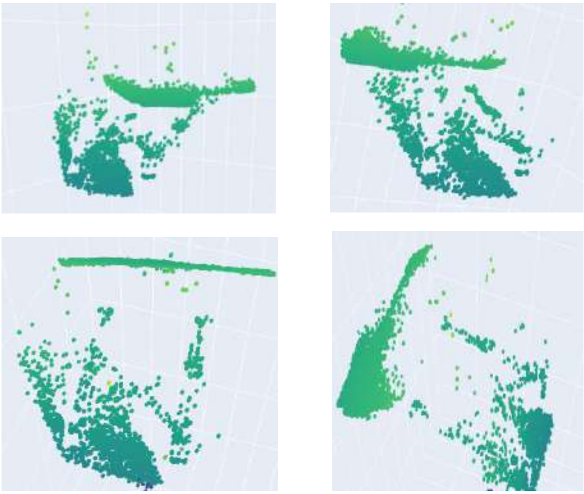
20 images (iphone camera)

Structure from Motion

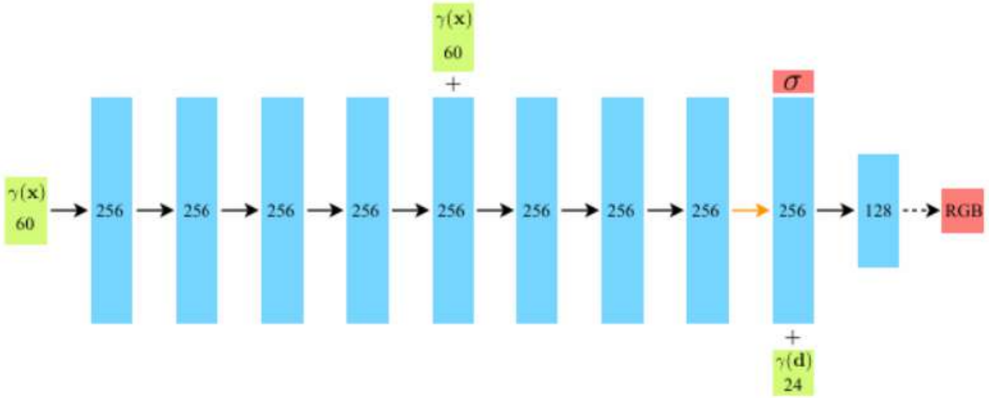
OpenCV

Key point detection and Feature matching (SIFT and FLANN)

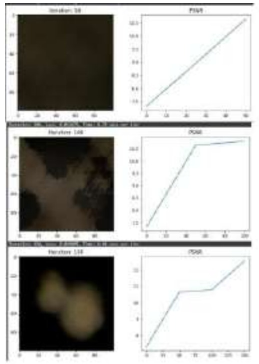
Triangulation

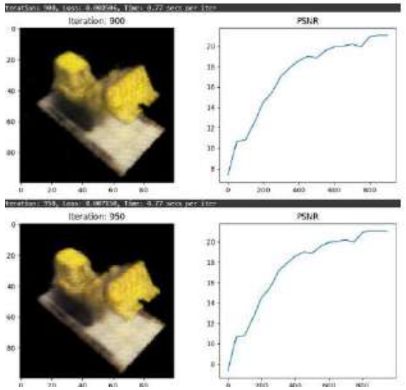


Input: Ray position and direction



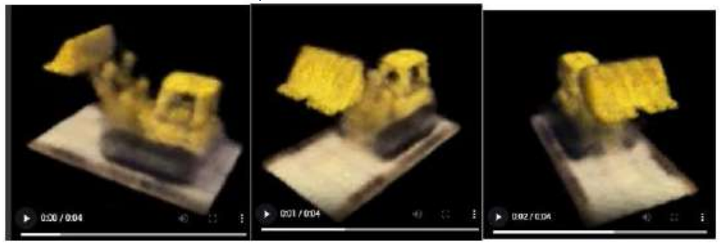
Training





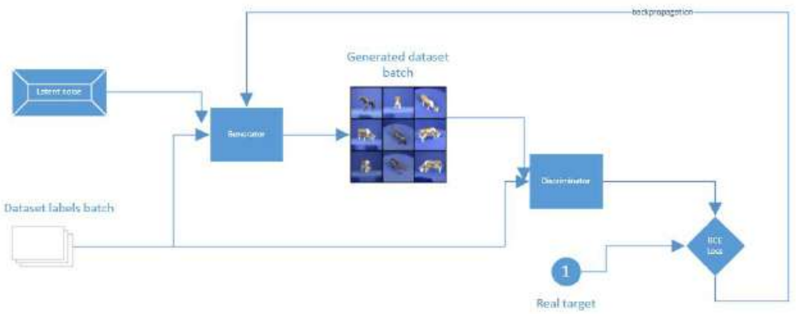
NeRF

Volume rendering

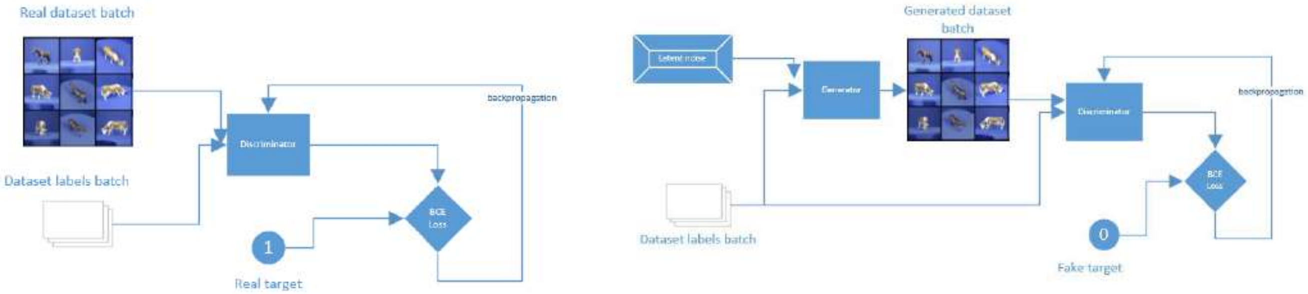


Projects

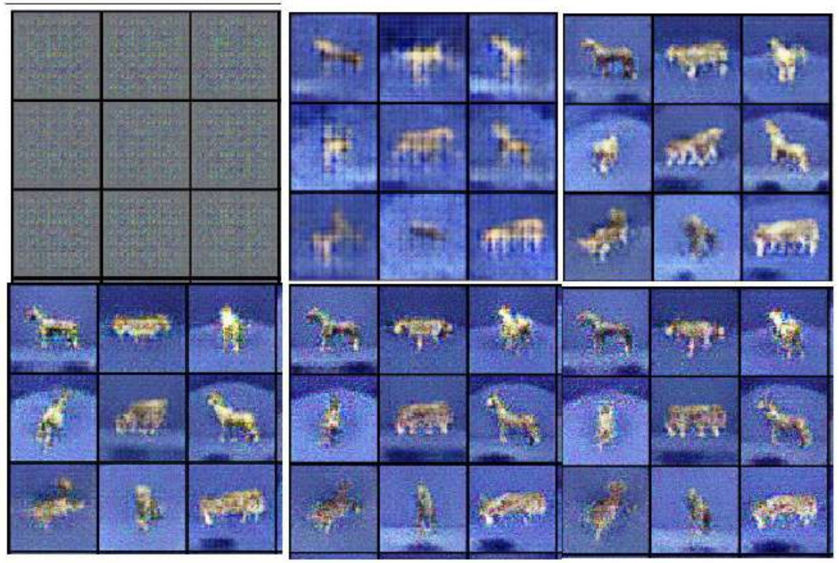
Generative Adversarial Networks for Data Augmentation



Generator Training



Discriminator Training



Training

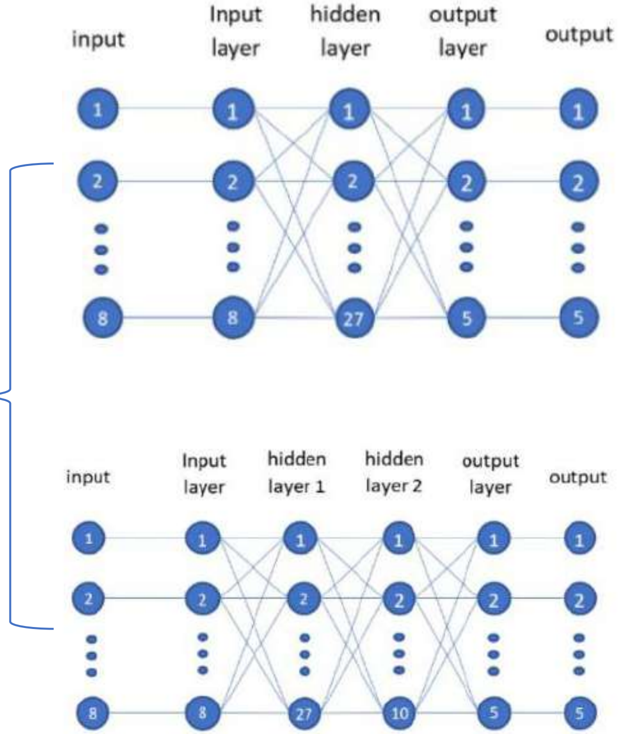
Real Images

Fake Images

Projects

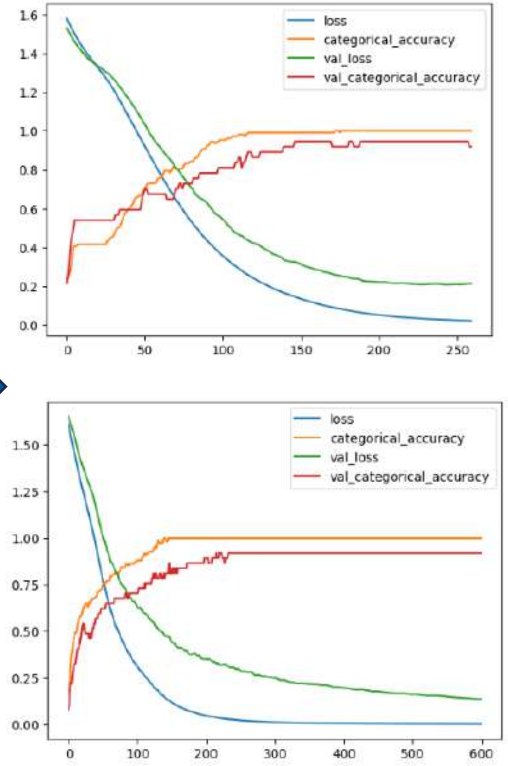
ANN for Autonomous vehicle Ultrasonic Multi-sensor system

Sensor's positions



2 ANN architectures

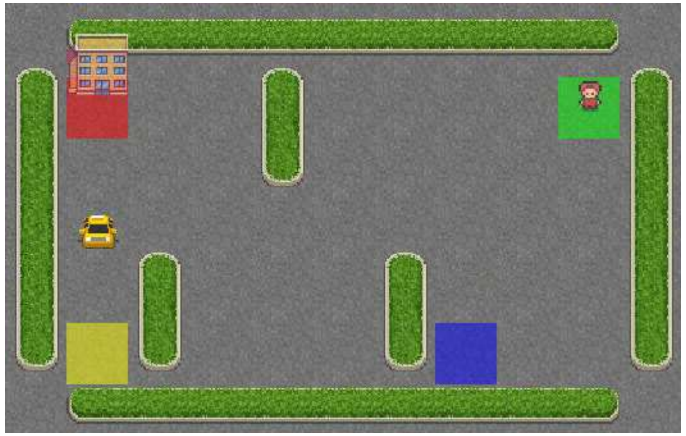
5 predictions:
- Stop/Braking
- Soft Braking
- Turn Left
- Turn right
- Go straight



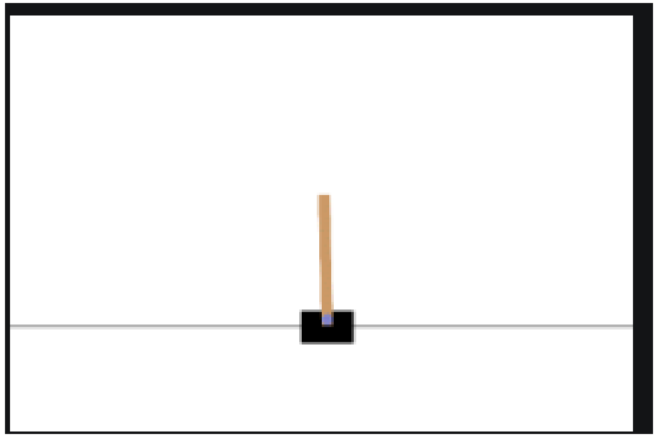
Performance of both networks

Projects

Reinforcement Learning mini projects



Q-Learning using Taxi-gym environment



DQN using Cart-Pole Gym environment

Trello Drone

Actions:
- Go forward
- Go back
- Turn clockwise (90 degrees)
- Turn anticlockwise (90 degrees)

RESNET CNN to get the features

Target plant

```
# Open drone video stream
videoUDP = 'udp://192.168.10.1:11111'
cap = cv2.VideoCapture(videoUDP)
```

Frame captured by the drone camera

Reward: Cosine similarity between both: Target features and frame features

Projects

Video Segmentation for a Tennis Clip

YOLO v5 to Segment
the Tennis ball

Kallman Filter using the YOLO predictions
to find the trajectory

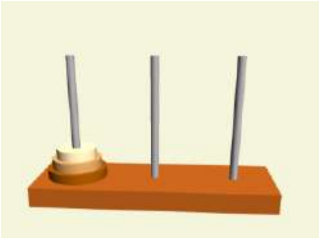
```
# Perform YOLOv5 detection on the frame
results = model(frame)

# Process YOLOv5 detections
detected_ball = None
for result in results.xywh[0]:  # Format: [x_center, y_center, width, height, confidence, class_id]
    x_center, y_center, w, h, conf, cls = result.tolist()
    if int(cls) == 32:  # Replace with the class ID for 'tennis ball' (check with your model)
        detected_ball = (int(x_center), int(y_center))
```

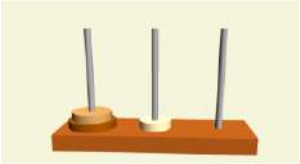
Find the speed

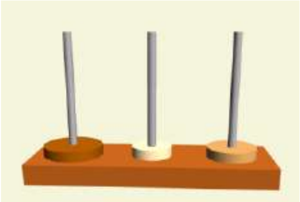
```
# Calculate speed (in pixels per frame)
if last_position:
    dt = 1 / fps  # Time difference between frames (in seconds)
    dx = predicted_x - last_position[0]
    dy = predicted_y - last_position[1]
    speed = np.sqrt(dx**2 + dy**2) / dt  # Speed in pixels per second
```

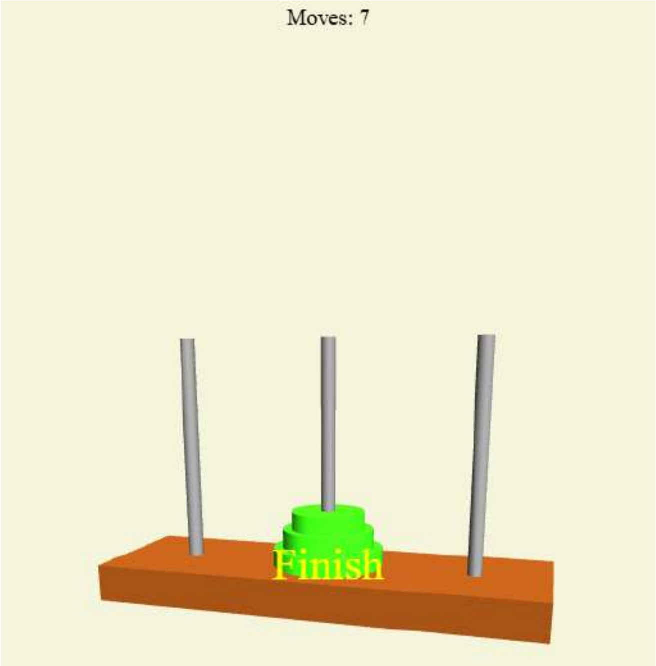
Projects

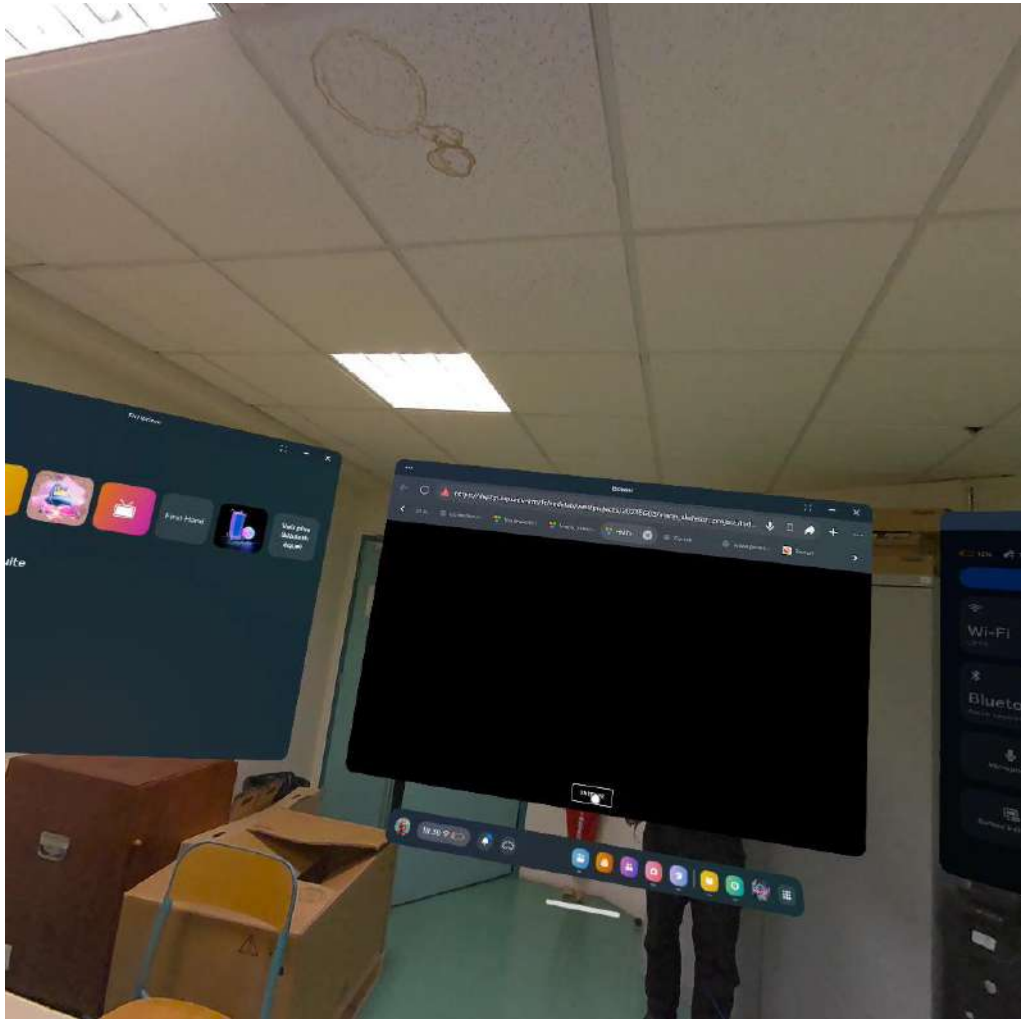
Tower of Hanoi XR implementation







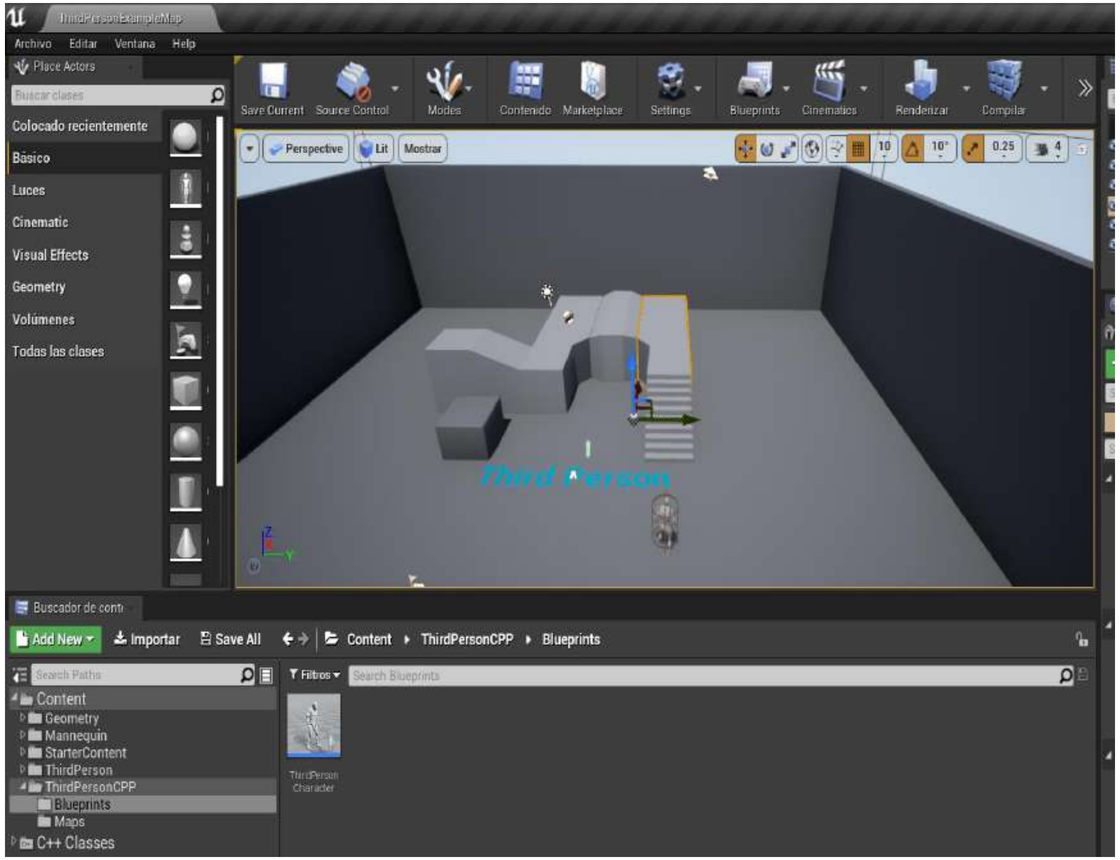




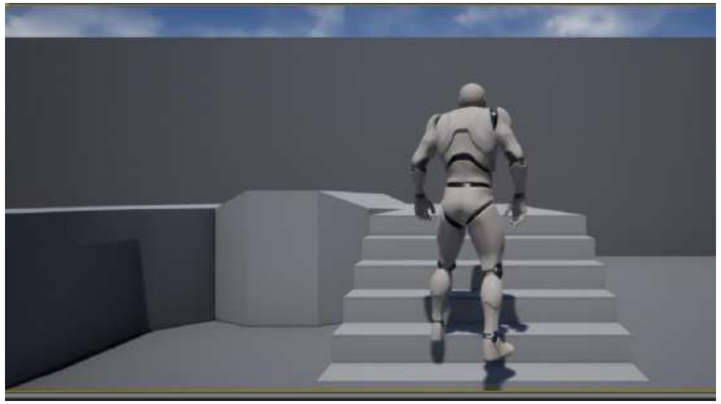
Meta Headsets

Job related experience

Basics of Unreal Engine



Using Blueprints



MERCI!