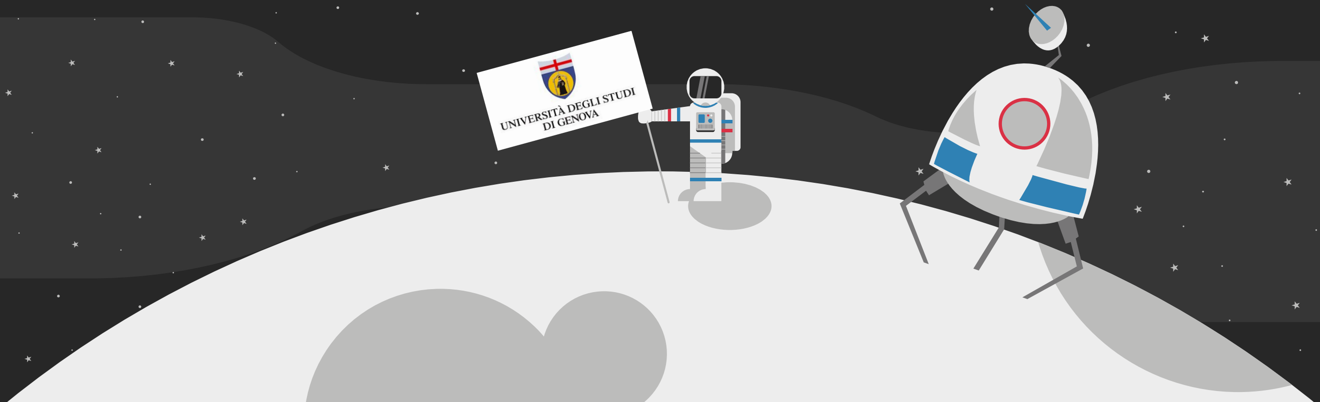
MOON LANDING

POSE ESTIMATION

Machine Learning Project



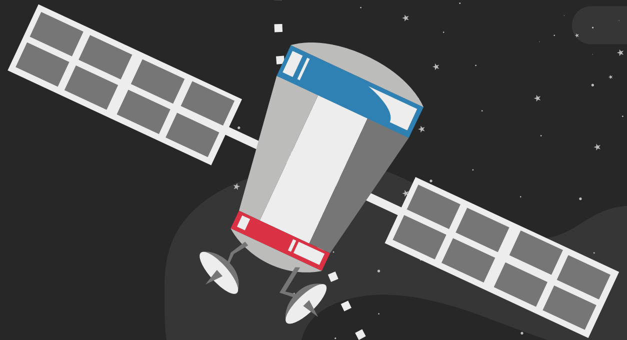
ECEM ISILDAR
S5430086

TABLE OF CONTENTS

01

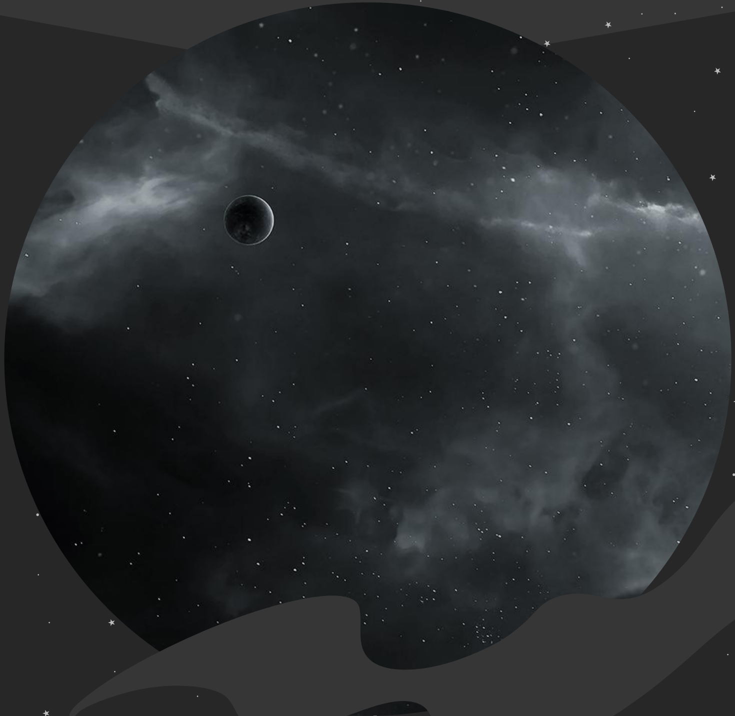
AIM

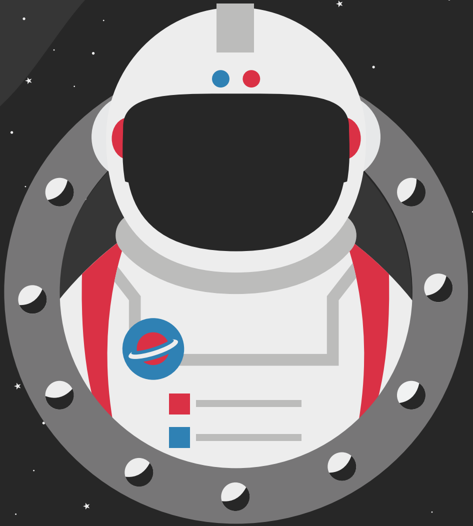
The goal of the project



What is the problem to be solved?

Using simulation data to train neural networks to estimate the pose of a rover's camera with respect to a known target object





02

DATASET

Learned pose estimation on the moon

ROBOT ON THE MOON

This dataset consists of RBG camera and depth images from a rover on a simulated lunar surface

- It already has labels for training
- It is already divided into 3 parts
  - 70% training
  - 20% validation
  - 10% test
- It has 5 different scenarios for different type of vehicle, I chose hopper dataset

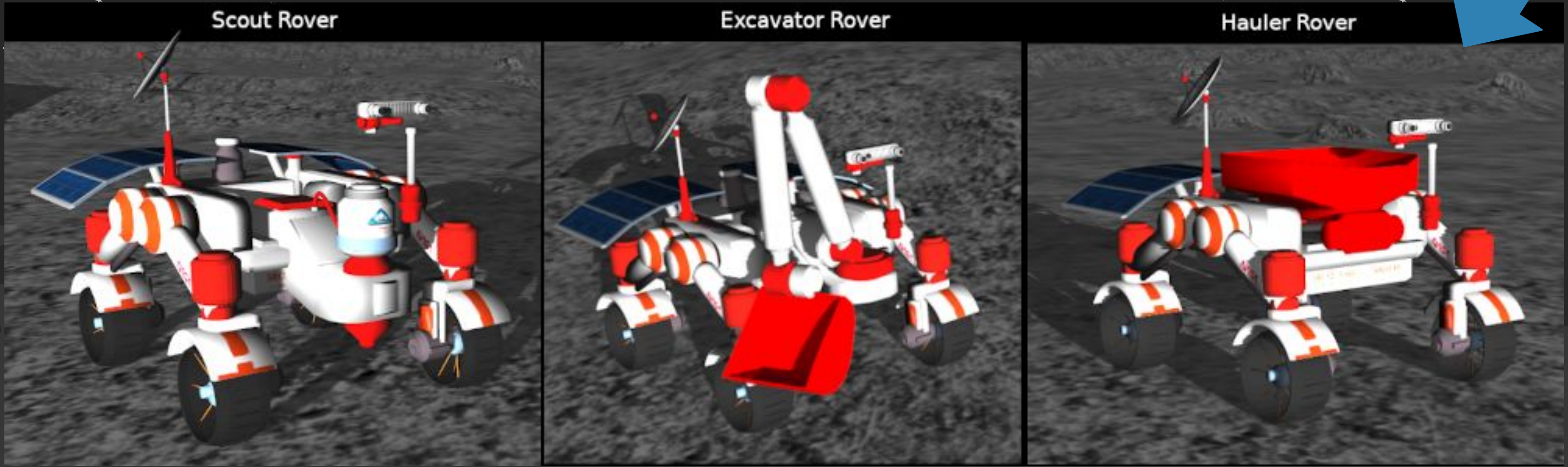
---

https://www.kaggle.com/datasets/louisburtz/learned-pose-estimation-robot-on-the-moon

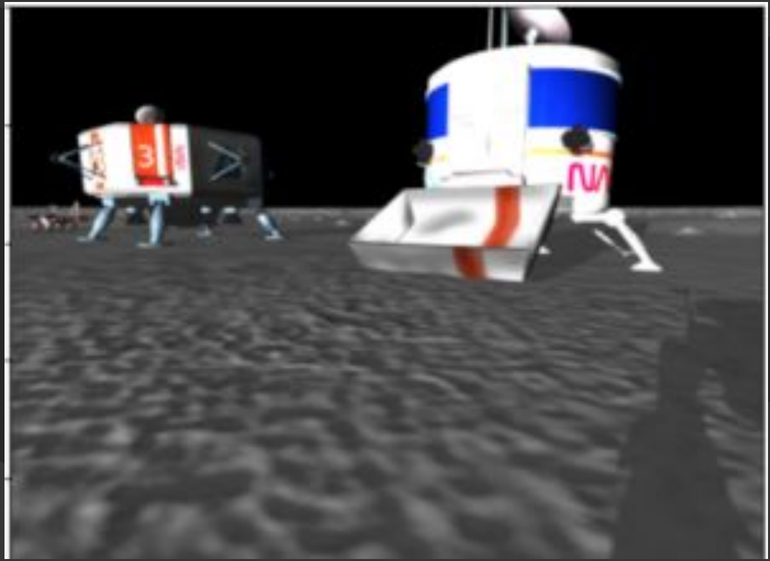
MORE DETAILS

This dataset contains 3 different types of rovers and each rover has a stereo camera mounted on a mast this is the sensor to create the RGBD images in the datasets.
The relative pose estimation problem is simplified from 6 DOF to just 3 variables:

- distance in polar coordinates
- theta
- yaw orientation



- There are two types of landers, both are always next to each other with fixed position and orientation in all dataset.
- The sun direction and terrain elements like craters, rocks, slopes, ground texture are identical in all datasets



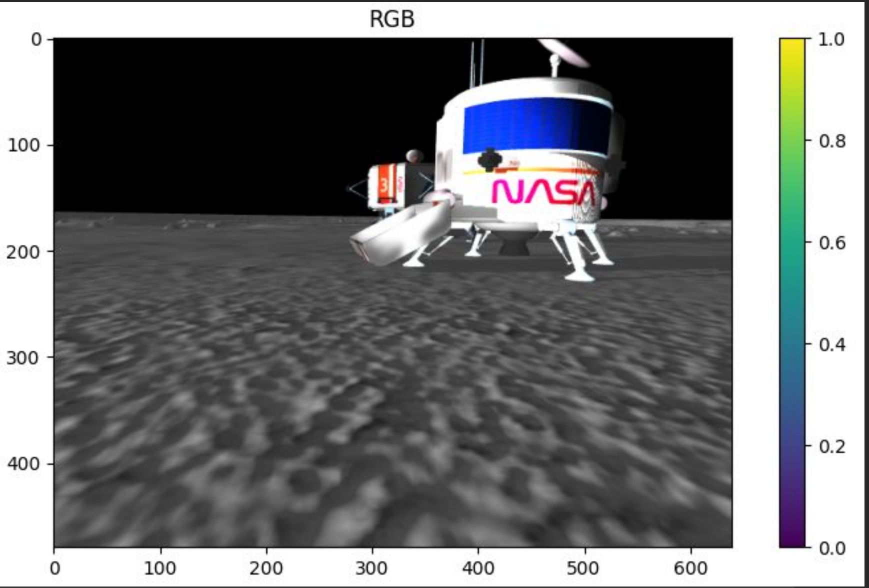
DATASET EXPLORATION

Label Distribution

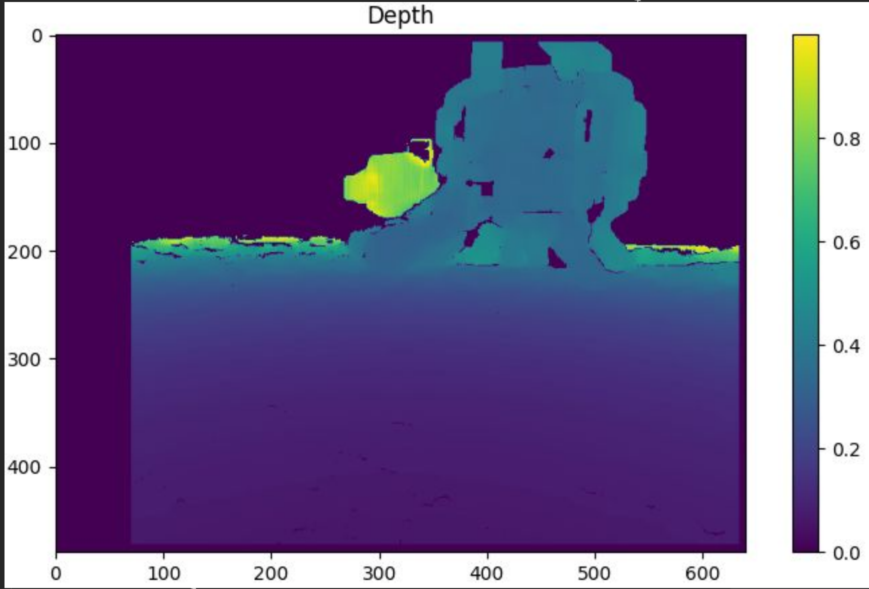
This plots show that uniform random distribution is provided for each parameter, within their bounds

Example Image from Dataset

RGB image



Depth image



03

ALGORITHM

Deep Learning Algorithm

THE MODEL

```
model = keras.Sequential(
[
    keras.Input(shape=input_shape),
    layers.experimental.preprocessing.Resizing(height, width),
    layers.Conv2D(256, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(16, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(8, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Flatten(),

    layers.Dense(6, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.1),
    layers.Dense(n_outputs, activation="linear"),
]
```

Corresponds to the dimensions of the input images (480, 640, 4).

Resizes the input images to the specified height (120) and width (160).

The series of Conv2D layers with varying numbers of filters and kernel sizes are used to extract features from the images.

After each set of convolutional layers, max-pooling is applied to reduce dimensions, which helps in reducing computational complexity and prevents overfitting.

This layer flattens the 2D feature maps into a 1D vector, preparing the data for the fully connected layers

THE MODEL

```
model = keras.Sequential(
[
    keras.Input(shape=input_shape),
    layers.experimental.preprocessing.Resizing(height, width),
    layers.Conv2D(256, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.Conv2D(128, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(64, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.Conv2D(32, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(16, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Conv2D(8, kernel_size=(3, 3), activation="relu", padding="same"),
    layers.MaxPooling2D(pool_size=pool_size),
    layers.Flatten(),

    layers.Dense(6, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dense(64, activation="relu"),
    layers.Dropout(0.1),
    layers.Dense(n_outputs, activation="linear"),
]
```

Image dimension (480, 640, 4),
Resized image (120, 160)

The sequence of Dense layers with relu activation is used to perform the final classification or regression task.

The final layer produces n_outputs number of outputs with a linear activation function, which might represents d, theta and yaw

Loss Functions

The overall pose_loss is a weighted combination of three components, where alpha and beta are weighting factors that determine the relative importance of orientation angle and orientation losses compared to the distance loss.

```
def pose_loss(y_true, y_pred):
    pose_loss = \
        ml_utils.distance_loss(y_true, y_pred) +  \
        alpha * ml_utils.theta_loss(y_true, y_pred) + \
        beta * ml_utils.orientation_loss(y_true, y_pred)
    return pose_loss

def theta_loss(y_true, y_pred):
    return alpha * ml_utils.theta_loss(y_true, y_pred)

def orientation_loss(y_true, y_pred):
    return beta * ml_utils.orientation_loss(y_true, y_pred)
```

alpha = 0.1
beta = 0.03
distance_loss = (distance_true - distance_pred)^2
orientation_loss = (orientation_true - orientation_pred)^2
theta_loss = (theta_true - theta_pred)^2

Callbacks

```
callbacks = [
    keras.callbacks.ReduceLROnPlateau(),
    keras.callbacks.EarlyStopping(
        monitor='val_loss',
        mode='min',
        patience=4,
        verbose=1,
        restore_best_weights=True
    )
]
```

This callback function reduces the learning rate (LR) when a metric has stopped improving, typically used when the model's training reaches a plateau.

This callback function stops training when a monitored metric has stopped improving, which helps prevent overfitting by halting the training process early if the model's performance on a validation set starts to degrade.

It monitors validation loss when it is minimum, it waits maximum 4 steps then it stops the training and stores the best weights

Model Compile

This function configures the neural network model for training by specifying the pose loss function (loss generated by combining 3 loss functions), optimizer (adam), and a set of metrics to evaluate the model's performance.

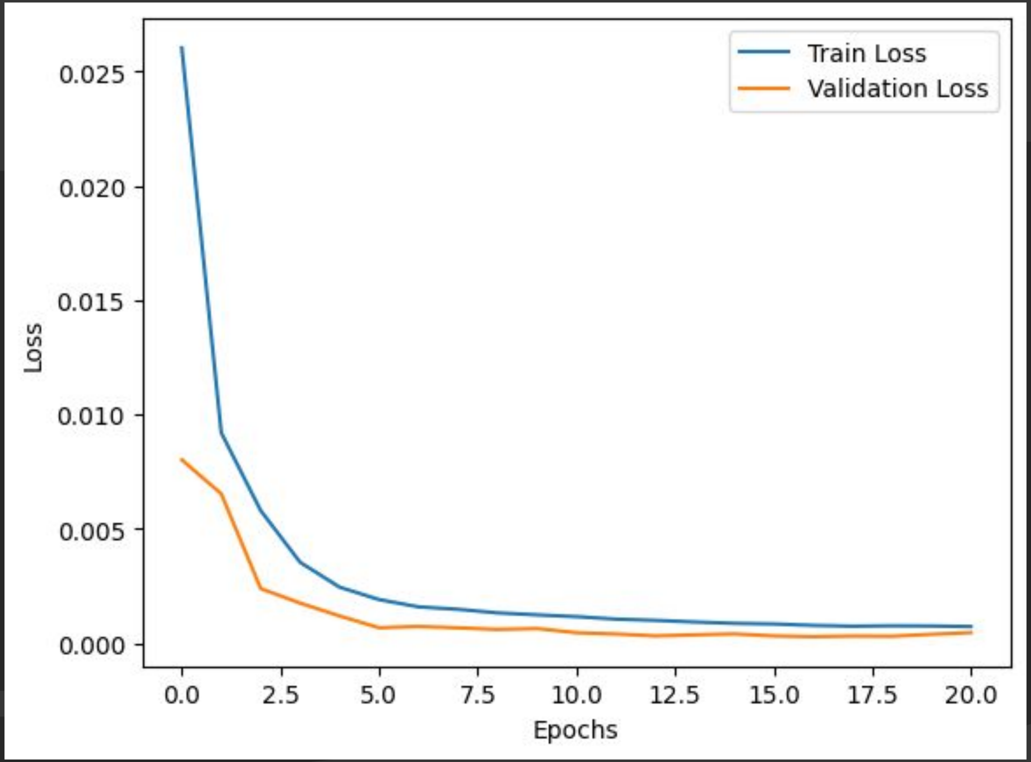
```
model.compile(
    loss=pose_loss,
    optimizer='adam',
    metrics=[
        ml_utils.distance_loss,
        theta_loss,
        orientation_loss,
        ml_utils.distance_diff,
        ml_utils.theta_diff,
        ml_utils.orientation_diff,
    ]
)
```

04

RESULTS

Is it a successful approach for learning?

LOSS CURVE INVESTIGATION



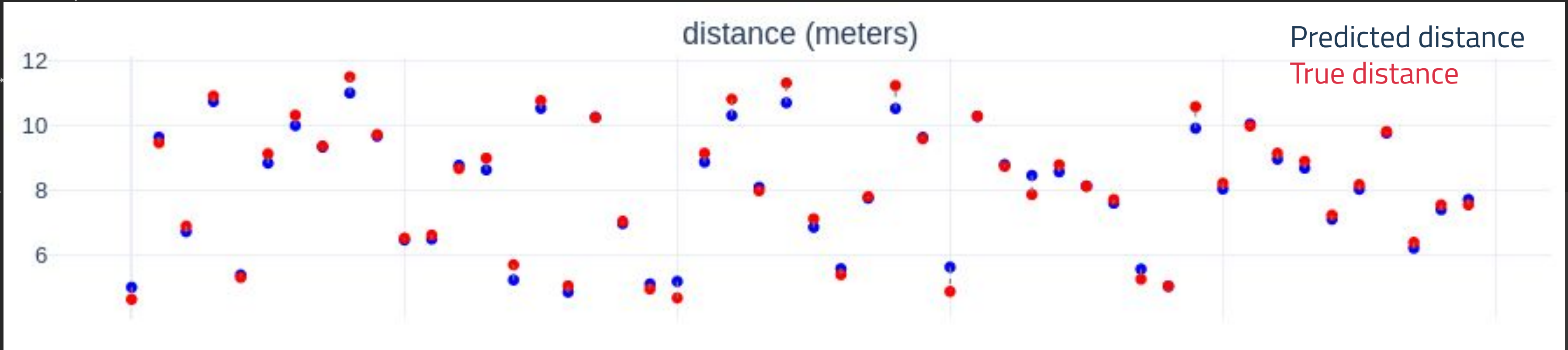
- Train and validation loss
- No overfitting

TEST EVALUATION

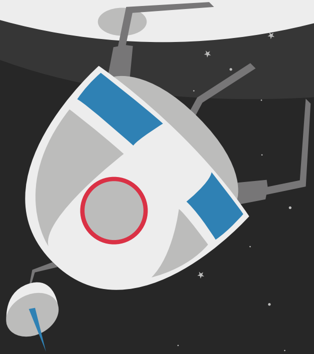
Training stopped at epoch 20 thanks to early stopping

Minimum Losses:

avg_position_diff = 0.193 meters



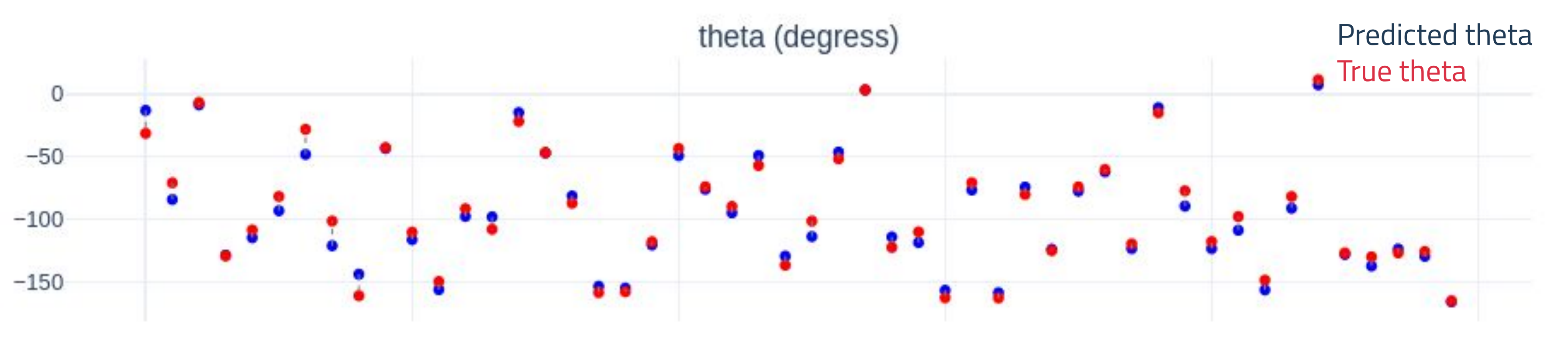
TEST EVALUATION



Training stopped at epoch 20 thanks to early stopping

Minimum Losses:

avg_theta_diff = 5.4 degrees

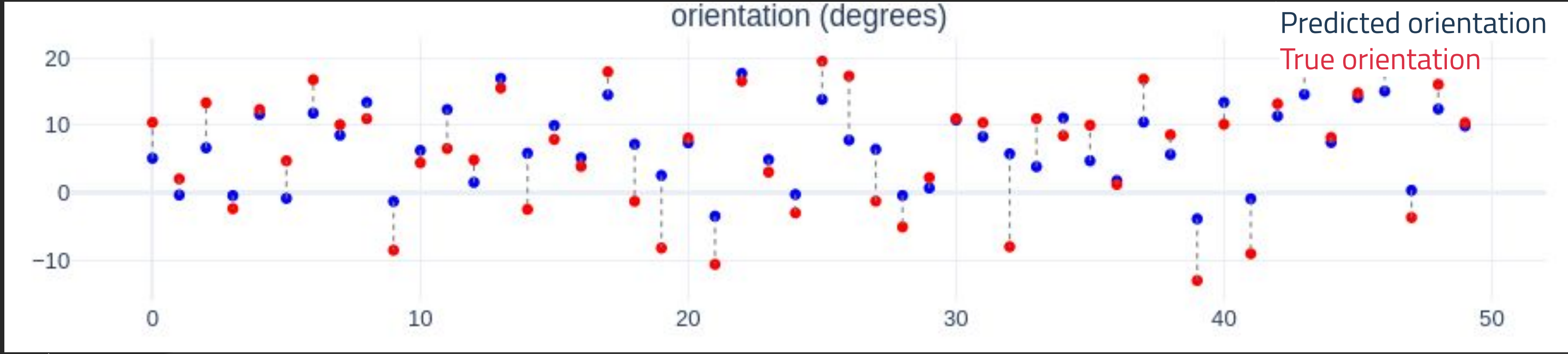


TEST EVALUATION

Training stopped at epoch 20 thanks to early stopping

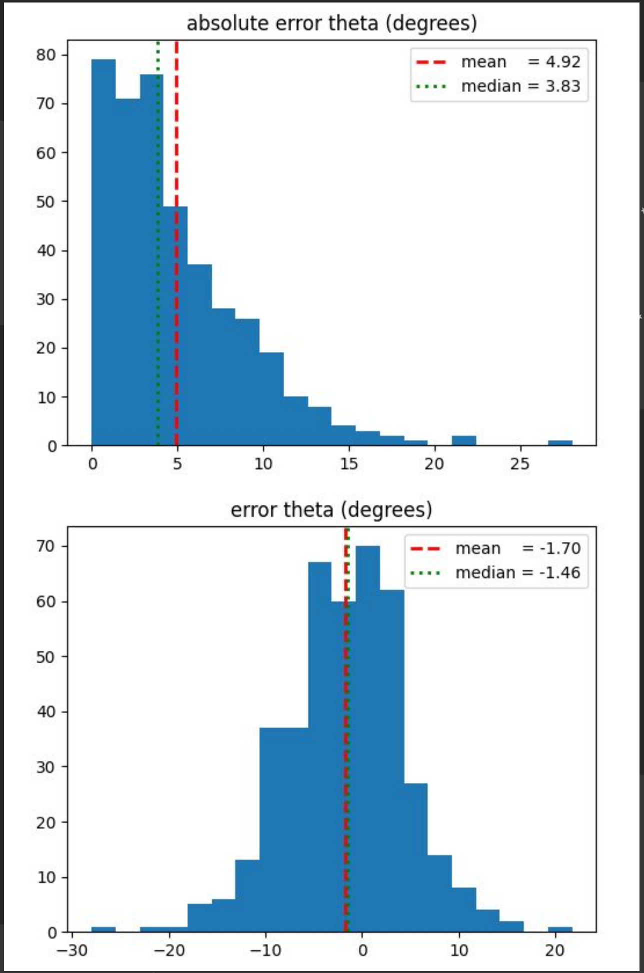
Minimum Losses:

avg_orientation_diff = 5.3 degrees



TEST EVALUATION

- Absolute error for theta degrees
- Model is not good enough to learn very small angles




THANK YOU!