Inference Program

Overview

This program performs inference on trajectory data using a Memory Neural Network. It loads trajectory data (beams and IMU), preprocesses it, constructs input samples, runs the model to predict velocity components, and saves the predictions to a CSV file.

Data and Workflow

- **Data Files:**
  - `beams_gt.csv`: Contains beam values with columns for time and four beams.
  - `IMU_*.csv` (IMU_trajectory14.csv for example): Contains IMU readings (acceleration and gyroscope data).
- **Workflow Steps:**
  1. **Data Loading & Synchronization:**
     - Reads the beams file and the first available IMU file from the trajectory folder.
     - Converts time stamps to strings and synchronizes data using common timestamps.

     Relevant Code:

```
def load_csv_files(traj_path):
    beams_df = pd.read_csv(os.path.join(traj_path, "beams_gt.csv"), na_values=[''])
    imu_files = [f for f in os.listdir(traj_path) if f.startswith("IMU_") and f.endswith(".csv")]
    if not imu_files:
        raise ValueError(f"No IMU file found in {traj_path}")
    imu_df = pd.read_csv(os.path.join(traj_path, imu_files[0]))

    beams_df['Time'] = beams_df['Time'].astype(str)
    imu_df['Time'] = imu_df['Time [s]'].astype(str)

    common_times = set(beams_df['Time']) & set(imu_df['Time'])
    if not common_times:
        raise ValueError("No common timestamps found between beams and IMU data.")

    beams_df = beams_df[beams_df['Time'].isin(common_times)].sort_values("Time").reset_index(drop=True)
    imu_df = imu_df[imu_df['Time'].isin(common_times)].sort_values("Time").reset_index(drop=True)

    return beams_df, imu_df
```

  2. **Preprocessing:**
     - Removes beam values that fall outside the valid range.
     - Fills missing beam values using a moving average over a defined window.

     Relevant Code for Removing Invalid Beams:

```
def remove_invalid_beam_values(beams_df, beam_cols=["b1", "b2", "b3", "b4"]):
    beams_df = beams_df.copy()
    for col in beam_cols:
        beams_df.loc[(beams_df[col] < VALID_BEAM_MIN) | (beams_df[col] > VALID_BEAM_MAX), col] = np.nan
    return beams_df
```

Computing SWITCH VALUE

SWITCH VALUE is a flag computed for each row:

It is set to 0 if 2 or more beam values (from b1 to b4) are missing.

Otherwise, it is set to 1

Relevant Code for Filling Missing Beams:

```
def fill_missing_beams(beams_df, beam_fill_window, beam_cols=["b1", "b2", "b3", "b4"]):
    filled = beams_df.copy()
    for i in range(beam_fill_window, len(filled)):
        for col in beam_cols:
            if pd.isna(filled.loc[i, col]):
                window = filled.loc[i - beam_fill_window:i - 1, col]
                if window.isna().all():
                    filled.loc[i, col] = filled[col].ffill().iloc[i - 1]
                else:
                    filled.loc[i, col] = window.mean()
    return filled
```

3. **Input Sample Construction:**

   - Constructs an input vector comprising:
     - Current beam values (4 values).
     - Five previous instances of beam values (5 × 4 values).
     - Current IMU readings (6 values).

   Relevant Code:

```
def construct_input_sample(filled_beams, imu_df, t, num_past_beam_instances, num_imu_instances):
    current_beams = filled_beams.loc[t, ["b1", "b2", "b3", "b4"]].values.astype(float)
    past_beams = []
    for i in range(1, num_past_beam_instances + 1):
        past_beams.extend(filled_beams.loc[t - i, ["b1", "b2", "b3", "b4"]].values.astype(float))
    imu_cols = ['ACC X [m/s^2]', 'ACC Y [m/s^2]', 'ACC Z [m/s^2]',
                'GYRO X [rad/s]', 'GYRO Y [rad/s]', 'GYRO Z [rad/s]']
    past_imu = []
    for i in range(num_imu_instances - 1, -1, -1):
        past_imu.extend(imu_df.loc[t - i, imu_cols].values.astype(float))
    input_vector = np.concatenate([current_beams, np.array(past_beams), np.array(past_imu)])
    return input_vector
```

4. **Model Initialization & Inference:**

   - Initializes the Memory Neural Network with parameters specified in the configuration.
   - Loads model weights from the checkpoint file.
   - Runs the model on each input sample to produce velocity predictions.

   Relevant Code:

```
def run_inference(trajectory, config, checkpoint_path, output_csv="inference_results.csv"):
    traj_path = os.path.join(DATA_DIR, trajectory)
    print(f"[INFO] Running inference on trajectory: {trajectory}")
    beams_df, imu_df = load_csv_files(traj_path)

    beams_df = remove_invalid_beam_values(beams_df)
    beam_fill_window = config.get("beam_fill_window", 5)
    filled_beams = fill_missing_beams(beams_df, beam_fill_window)

    first_valid = find_first_valid_index(filled_beams, imu_df,
                                         config["num_past_beam_instances"],
                                         config["num_imu_instances"])
    if first_valid is None:
        raise ValueError("No valid starting index found in trajectory " + trajectory)

    filled_beams = filled_beams.iloc[first_valid:].reset_index(drop=True)
    imu_df = imu_df.iloc[first_valid:].reset_index(drop=True)
    time_series = filled_beams["Time"].tolist()

    num_samples = len(filled_beams)
    inputs = []
    valid_indices = []
    start_t = max(config["num_past_beam_instances"], config["num_imu_instances"] - 1)
    for t in range(start_t, num_samples):
        try:
            sample = construct_input_sample(filled_beams, imu_df, t,
                                            config["num_past_beam_instances"],
                                            config["num_imu_instances"])
            inputs.append(sample)
            valid_indices.append(t)
        except Exception as e:
            print(f"[WARNING] Skipping index {t}: {e}")
            continue

    if len(inputs) == 0:
        raise ValueError("No valid inference samples in trajectory " + trajectory)

    inputs = np.array(inputs)
    input_size = inputs.shape[1]

    model = MemoryNeuralNetwork(number_of_input_neurons=input_size,
                                number_of_hidden_neurons=config["hidden_neurons"],
                                number_of_output_neurons=3,
                                dropout_rate=config["dropout_rate"],
                                learning_rate=config["learning_rate"],
                                learning_rate_2=config["learning_rate_2"],
                                lipschitz_constant=config["lipschitz_constant"])

    model.load_state_dict(torch.load(checkpoint_path, map_location=model.device))
    model.eval()

    predictions = []
    for idx, sample in enumerate(inputs):
        sample_time = filled_beams.loc[valid_indices[idx], "Time"]
        x = torch.tensor(sample, dtype=torch.float32, device=model.device).unsqueeze(0).unsqueeze(0)
        with torch.no_grad():
```

```
            y_pred = model(x).squeeze().view(-1)
        pred_velocities = y_pred.cpu().numpy().tolist()
        predictions.append({
            "Time": sample_time,
            "V_X": pred_velocities[0],
            "V_Y": pred_velocities[1],
            "V_Z": pred_velocities[2]
        })

    output_path = os.path.join(os.getcwd(), output_csv)
    pd.DataFrame(predictions).to_csv(output_path, index=False)
    print(f"[INFO] Inference complete. Predictions saved to {output_path}")
```

5. **Output Generation:**

   - After processing each input sample, the predictions are combined with their corresponding timestamps along with a switch value.

   Relevant Code:

```
# Save predictions to CSV
output_path = os.path.join(os.getcwd(), output_csv)
pd.DataFrame(predictions).to_csv(output_path, index=False)
print(f"[INFO] Inference complete. Predictions saved to {output_path}")
```

Configuration (MNN.json)

The configuration file contains the following parameters:

```
{
  "beam_fill_window": 5, // The number of past instances of DVL used to calculate the moving average
  "num_past_beam_instances": 3, // Number of past DVL instances used as inputs to the Network
  "num_imu_instances": 1, // Number of total IMU instances used as inputs to the Network
  "hidden_neurons": 100
}
```

Execution

1. **Prepare the Trajectory Data:**

   - Place the trajectory folder (e.g., `Trajectory15`) containing the required CSV files in the designated data directory.

2. **Run the Inference Script:**

   - Execute the script from the command line:

```
python Inference.py
```

   - The program will load, preprocess, and synchronize the data, construct input samples, run the Memory Neural Network for inference, and generate the output CSV file.