POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

AI Driving Classification 2023/2024

Team Members:

Alberto Manuel de Matos Pingo - 2202145

João Pedro Quintela de Castro - 2201781

First Approach - Bidirectional LSTM

Libraries

In [19]:
```
import os
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import tensorflow as tf

from tensorflow import keras
from keras.models import Sequential, load_model
from keras.layers import LSTM, Dense, Dropout,Conv1D,Flatten,Bidirectional,BatchNormalization
from keras.callbacks import EarlyStopping, ReduceLROnPlateau,ModelCheckpoint
import keras.backend as K

from sklearn import metrics
from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import recall_score
from sklearn.metrics import accuracy_score
from sklearn.metrics import hamming_loss, jaccard_score, coverage_error, label_ranking_loss

import folium

import seaborn as sns

from folium.plugins import MarkerCluster
from folium.plugins import FastMarkerCluster
from folium import plugins

import warnings
warnings.filterwarnings('ignore')
```

Directories

In [46]:
```
import os

# Make auxiliar folders
if not os.path.exists('runtime_saves'):
    os.makedirs('runtime_saves')
if not os.path.exists('runtime_saves/models'):
    os.makedirs('runtime_saves/models')
if not os.path.exists('runtime_saves/data'):
    os.makedirs('runtime_saves/data')
if not os.path.exists('runtime_saves/visual'):
    os.makedirs('runtime_saves/visual')

dir_current = os.getcwd()
dir_root = os.path.abspath(os.path.join(dir_current, os.pardir, os.pardir))

dir_datasets = os.path.join(dir_root, 'datasets')
```

```
# Datasets
    # IPL-Dataset
dataset_Abrantes = os.path.join(dir_root, 'datasets', 'Abrantes-Leiria.csv')
dataset_DatasetAll = os.path.join(dir_root, 'datasets', 'Dataset-All.csv')
    # UAH
dataset_UAH_dir = os.path.join(dir_root, 'datasets', 'UAH-DRIVESET-v1', 'UAH-Processed')

print(f'Root directory: {dir_root}')
print(f'Datasets directory: {dir_datasets}')
print(f'Dataset directory: {dataset_Abrantes}')
```

```
Root directory: c:\codeUni\ProjetoInformatico\CoEProject-AI-DrivingClassification
Datasets directory: c:\codeUni\ProjetoInformatico\CoEProject-AI-DrivingClassification\datasets
Dataset directory: c:\codeUni\ProjetoInformatico\CoEProject-AI-DrivingClassification\datasets\Abrantes-Leiria.cs
v
```

AUX FUNCTIONS

In [47]:

```
def save_manovers_positions_to_csv_file(gps_positions, manovers, filename):
  output = np.zeros_like(gps_positions)

  # Iterate through the elements of arr2
  for i in range(len(manovers)):
    # Check if the element in arr2 is 1
    if manovers[i] == 1:
      # Copy the corresponding values from arr1 to the output array
      output[i] = gps_positions[i]

  output = output[~np.all(output == 0, axis=1)]

  filename = 'runtime_saves/data/maneuver_' + filename

  np.savetxt(filename, output, delimiter=',', fmt='%.9f')

def separate_positives_negatives(data):
  # Ensure the input is converted to a NumPy array for easier manipulation
  data = np.array(data)

  # Create two empty arrays to store positive and negative values
  positives = np.zeros_like(data)
  negatives = np.zeros_like(data)

  # Use boolean indexing to separate positive and negative values
  positives[data > 0] = data[data > 0]
  negatives[data < 0] = -data[data < 0]

  # Combine the positive and negative values into a single 2D array
  return (positives, negatives)

def normalize_between_0_and_max(data):
  max_value = np.max(data)
  return data / max_value

def normalize_between_0_and_max_v2(data, max_value):
  return data / max_value

def split_train_test(data, test_size=0.2):
  # Check if test_size is between 0 and 1
  if test_size < 0 or test_size > 1:
    raise ValueError("test_size must be between 0 and 1.")

  # Get the number of samples
  num_samples = data.shape[0]

  # Calculate the number of samples for each set
  train_size = int(num_samples * (1 - test_size))
  test_size = num_samples - train_size

  # Randomly shuffle the data for better splitting (optional)
  #np.random.shuffle(data)

  # Split the data into training and test sets
  train_data = data[:train_size]
  test_data = data[train_size:]

  return train_data, test_data

def y_classification(data, threshold):
  classification = np.zeros_like(data, dtype=int)  # Initialize output array

  for col in range(0, 12):  # Loop through each column
```

```
    max_value = np.max(data[:, col])
    threshold_pos = max_value * threshold
    classification[:, col] = np.where(data[:, col] >= threshold_pos, 1, 0)

  return classification

def max_of_vectors(vec1, vec2, vec3, vec4, vec5, vec6):
  # Combine all vectors into a single array
  all_vectors = np.array([vec1, vec2, vec3, vec4, vec5, vec6])

  # Find the maximum value in the array
  max_value = np.max(all_vectors)

  return max_value

def has_one(data):
  """
  This function receives a numpy array and returns a new array
  with 1 if the correspondent row of input array has at least one cellule with 1.
  In other case the cellule is 0.

  Args:
      data: A numpy array of shape (n, 12) with 0 or 1 values in each cell.

  Returns:
      A numpy array of shape (n, 1) with 1s where the corresponding row in data has at least one 1, and 0s othe
  """
  # We sum each row, and any value greater than zero indicates at least one 1 in that row
  return np.sum(data, axis=1)[:, np.newaxis] > 0
```

DATA PREPROCESSING

Data Structure

Accelerometer (m/s²): Acceleration along the each axis.

- **accelerometerXAxis**
- **accelerometerZAxis**
- **accelerometerYAxis**

Gyroscope (°/s): Angular velocity along the each axis.

- **gyroscopeXAxis**
- **gyroscopeYAxis**
- **gyroscopeZAxis**

GPS Coordinates (°):

- **Latitude**
- **Longitude**

In [48]:

```
# Load the dataset into a DataFrame
df = pd.read_csv(dataset_Abrantes)
#df = pd.read_csv(dataset_DatasetAll)

acelX = df['accelerometerXAxis']
acelY = df['accelerometerYAxis']
acelZ = df['accelerometerZAxis']

gyrX = df['gyroscopeXAxis']
gyrY = df['gyroscopeYAxis']
gyrZ = df['gyroscopeZAxis']

latitude = df['latitude']
longitude = df['longitude']
```

Separate data by maneuver

We identify different manovers based on the **Acceleration and Gyroscope data**.

Accelerometer:

- X - Curves
- Y - Acceleration and braking
- Z - Vertical acceleration - Uphill and downhill

Gyroscope:

- X - Longitudinal tilt - Uphill and downhill
- Y - Lateral tilt
- Z - Curves

In [24]:
```
maneuvers = [
    'Left Turn (Acc X+)', 'Right Turn (Acc X-)',
    'Acceleration (Acc Y+)', 'Braking (Acc Y-)',
    'Ascent (Acc Z+)', 'Descent (Acc Z-)',
    'Left Lateral Tilt (Gyro X+)', 'Right Lateral Tilt (Gyro X-)',
    'Forward Tilt (Gyro Y+)', 'Backward Tilt (Gyro Y-)',
    'Left Turn (Gyro Z+)', 'Right Turn (Gyro Z-)'
]

maneuvers_short = [
    'LT (Acc X+)', 'RT (Acc X-)',
    'Acc (Acc Y+)', 'Brk (Acc Y-)',
    'Asc (Acc Z+)', 'Des (Acc Z-)',
    'Lat L (Gyro X+)', 'Lat R (Gyro X-)',
    'Tilt F (Gyro Y+)', 'Tilt B (Gyro Y-)',
    'LT (Gyro Z+)', 'RT (Gyro Z-)'
]
```

In [50]:
```
# Curves (based on Acceleration along X-axis)
turnRightX, turnLeftX = separate_positives_negatives(acelX)
# Acceleration and braking
accelY, breakY = separate_positives_negatives(acelY)
# Vertical acceleration - ascent and descent
positiveZ, negativeZ = separate_positives_negatives(acelZ)

# Lateral tilts - right and left
gyrPositiveX, gyrNegativeX = separate_positives_negatives(gyrX)

# Forward and backward tilts
gyrPositiveY, gyrNegativeY = separate_positives_negatives(gyrY)

# Curves (based on Gyro along Z-axis)
gyrPositiveZ, gyrNegativeZ = separate_positives_negatives(gyrZ)

turnRightX.shape
```

Out[50]:
```
(35129,)
```

Normalize Data

This process **helps the model performance** by rescaling specified columns of the dataset to a range between 0 and a max value.

We identify the max value of the **original 3 axis** of the accelerometer and the **3 axis** of the gyroscope.

In [51]:
```
max_accel = max_of_vectors(turnRightX, turnLeftX, accelY, breakY, positiveZ, negativeZ)
max_gyr = max_of_vectors(gyrPositiveX, gyrNegativeX, gyrPositiveY, gyrNegativeY, gyrPositiveZ, gyrNegativeZ)

turnRightXn = normalize_between_0_and_max_v2(turnRightX, max_accel)
turnLeftXn = normalize_between_0_and_max_v2(turnLeftX, max_accel)
accelYn = normalize_between_0_and_max_v2(accelY, max_accel)
breakYn = normalize_between_0_and_max_v2(breakY, max_accel)
positiveZn = normalize_between_0_and_max_v2(positiveZ, max_accel)
negativeZn = normalize_between_0_and_max_v2(negativeZ, max_accel)
gyrPositiveXn = normalize_between_0_and_max_v2(gyrPositiveX, max_gyr)
gyrNegativeXn = normalize_between_0_and_max_v2(gyrNegativeX, max_gyr)
gyrPositiveYn = normalize_between_0_and_max_v2(gyrPositiveY, max_gyr)
gyrNegativeYn = normalize_between_0_and_max_v2(gyrNegativeY, max_gyr)
gyrPositiveZn = normalize_between_0_and_max_v2(gyrPositiveZ, max_gyr)
gyrNegativeZn = normalize_between_0_and_max_v2(gyrNegativeZ, max_gyr)
```

Concatenate Data

This process concatenates all columns into a single DataFrame after the previous preprocessing steps.

In [52]:
```
x = np.array(list(zip(turnRightXn, turnLeftXn, accelYn, breakYn, positiveZn, negativeZn, gyrPositiveXn, gyrNega

x.shape
```

Out[52]:
```
(35129, 12)
```

Labelling Data

The labelling is done considering:

- The **Max value** of each column
- An **Adjustable threshold** between 0 and 1.

The product of this maximum value and the threshold establishes a reference point that indicates the intensity of the maneuver.

- If the data value is greater than or equal to the reference point, it will be classified as 1 (aggressive).

- If the data value is less than the reference point, it will be classified as 0 (non-aggressive).

In [53]:
```
y = y_classification(x, 0.3)
print (np.sum(y, axis=0))

print(y)

np.savetxt('runtime_saves/data/Y.csv', y, delimiter=',', fmt='%.0i')
```

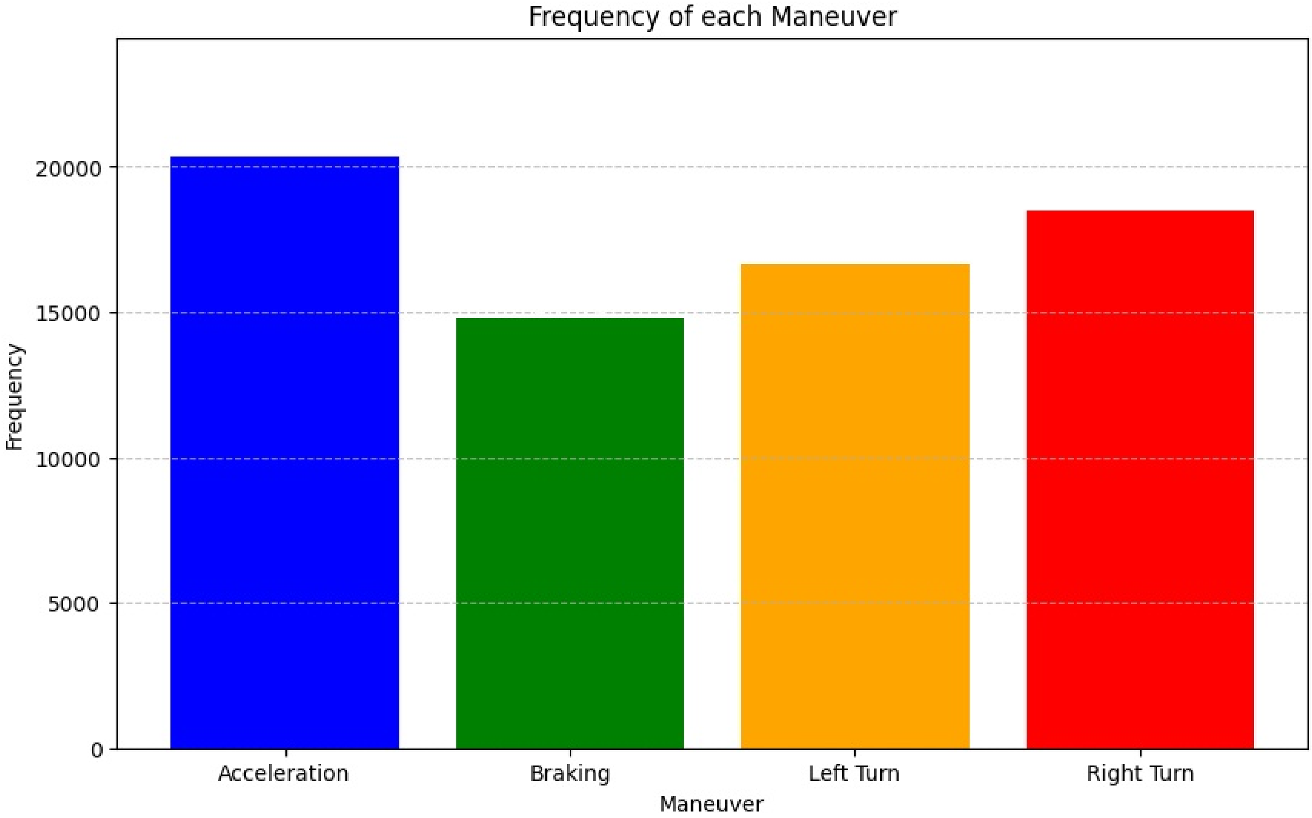
```
[ 945  836  714  157  259  115  421  687  719  712 1144  375]
[[0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 ...
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]
 [0 0 0 ... 0 0 0]]
```

Maneuvers Statistics

In [54]:
```
def count_maneuvers(accelY, breakY, turnLeftXn, turnRightXn):
    maneuvers = {
        'Acceleration': np.count_nonzero(accelY),
        'Braking': np.count_nonzero(breakY),
        'Left Turn': np.count_nonzero(turnLeftXn),
        'Right Turn': np.count_nonzero(turnRightXn)
    }

    return maneuvers

#calcula a disntancia entre dois pontos
def haversine(lat1, lon1, lat2, lon2):
    R = 6371000  # +- raio da terra em metros
    phi_1 = math.radians(lat1)
    phi_2 = math.radians(lat2)
    delta_phi = math.radians(lat2 - lat1)
    delta_lambda = math.radians(lon2 - lon1)

    a = math.sin(delta_phi / 2.0) ** 2 + math.cos(phi_1) * math.cos(phi_2) * math.sin(delta_lambda / 2.0) ** 2
    c = 2 * math.atan2(math.sqrt(a), math.sqrt(1 - a))

    return R * c

def total_distance(lat, lon):
    distance = 0.0
    for i in range(1, len(lat)):
        distance += haversine(lat[i], lon[i], lat[i - 1], lon[i - 1])
    return distance

print(f'Total distance: {total_distance(latitude, longitude) / 1000:.2f} km')

print(f'Total # of maneuvers: {count_maneuvers(accelY, breakY, turnLeftXn, turnRightXn)}')

maneuvers_count = count_maneuvers(accelY, breakY, turnLeftXn, turnRightXn)

maneuvers_labels = list(maneuvers_count.keys())
maneuvers_quantidades = list(maneuvers_count.values())

colors = ['blue', 'green', 'orange', 'red']

plt.figure(figsize=(10, 6))
bars = plt.bar(maneuvers_labels, maneuvers_quantidades, color=colors)

for bar, quantity in zip(bars, maneuvers_quantidades):
    plt.text(bar.get_x() + bar.get_width() / 2, bar.get_height() - 0.1, quantity,
             ha='center', va='bottom', color='white', fontsize=12)

plt.xlabel('Maneuver')
plt.ylabel('Frequency')
```

```python
plt.title('Frequency of each Maneuver')
plt.ylim(0, max(maneuvers_quantidades) * 1.2)
plt.grid(axis='y', linestyle='--', alpha=0.7)
plt.show()
```

```
Total distance: 84.11 km
Total # of maneuvers: {'Acceleration': 20354, 'Braking': 14775, 'Left Turn': 16657, 'Right Turn': 18472}
```



Export maneuvers

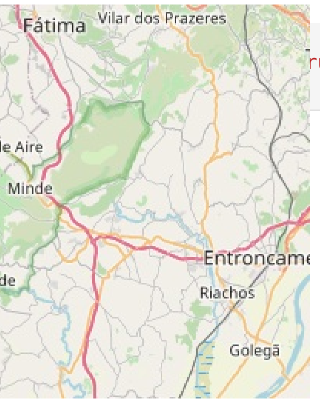
Can be used on Google Maps to visualize the maneuvers.

In [55]:
```python
positions = np.array(list(zip(latitude, longitude)))
maneuvers_accelY = y[:, 2]
maneuvers_breakY = y[:, 3]
maneuvers_turnRightXn = y[:, 0]
maneuvers_turnLeftXn = y[:, 1]
gyrPositiveZn = y[:, 10]
gyrNegativeZn = y[:, 11]
save_manovers_positions_to_csv_file(positions, maneuvers_accelY, "accelY.csv")
save_manovers_positions_to_csv_file(positions, maneuvers_breakY, "breakY.csv")
save_manovers_positions_to_csv_file(positions, maneuvers_turnRightXn, "turnRightX.csv")
save_manovers_positions_to_csv_file(positions, maneuvers_turnLeftXn, "turnLeftX.csv")
save_manovers_positions_to_csv_file(positions, gyrPositiveZn, "gyrPositZ.csv")
save_manovers_positions_to_csv_file(positions, gyrNegativeZn, "gyrNegZ.csv")
```

Folium Map maneuvers

In [56]:
```python
map = folium.Map(location=[np.mean(latitude), np.mean(longitude)], zoom_start=10)


marker_cluster = MarkerCluster().add_to(map)

for i in range(len(positions)):
    if maneuvers_accelY[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='blue'),popup=f'Acceleration')
    if maneuvers_breakY[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='red'),popup=f'Braking').add_t
    if maneuvers_turnRightXn[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='green'),popup=f'Left Turn (Ac
    if maneuvers_turnLeftXn[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='orange'),popup=f'Right Turn (

    if gyrNegativeZn[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='black'),popup=f'Left Turn (Gy
    if gyrPositiveZn[i] == 1:
        folium.Marker([positions[i][0], positions[i][1]], icon=folium.Icon(color='purple'),popup=f'Rigth Turn ((
```

```
runtime_saves/maneuvers.html')
```



Split Dataset for Model Training

This section splits the dataset into training, test, and validation sets

- 75% Training
- 25% Test

In [57]:
```python
# Split into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state=42)

#shape
print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

unique, counts = np.unique(y_train, return_counts=True)
print(dict(zip(unique, counts)))
```

```
(26346, 12) (26346, 12)
(8783, 12) (8783, 12)
{0: 310859, 1: 5293}
```

Create input tensor data

In [58]:
```python
X_train = X_train.reshape(X_train.shape[0], 1, X_train.shape[1])
X_test = X_test.reshape(X_test.shape[0], 1, X_test.shape[1])

print(X_train.shape, y_train.shape)
print(X_test.shape, y_test.shape)

# Save
np.save('runtime_saves/data/X_train.npy', X_train)
np.save('runtime_saves/data/X_test.npy', X_test)
np.savetxt('runtime_saves/data/y_train.csv', y_train, delimiter=',', fmt='%.0i')
np.savetxt('runtime_saves/data/y_test.csv', y_test, delimiter=',', fmt='%.0i')

np.savetxt('runtime_saves/data/X_train.csv', X_train.reshape(X_train.shape[0], X_train.shape[2]), delimiter=','
np.savetxt('runtime_saves/data/X_test.csv', X_test.reshape(X_test.shape[0], X_test.shape[2]), delimiter=',', fm
```

```
(26346, 1, 12) (26346, 12)
(8783, 1, 12) (8783, 12)
```

MODEL ARCHITECTURE - Bidirectional LSTM

Architecture:

Input -> Bidirectional LSTM - BN -> Dropout -> Bidirectional LSTM - BN -> Dropout -> Dense - BN -> Dropout -> Dense - Output

1. **Input Layer**

   - The input layer expects sequences with shape `(timesteps, features)`, where `timesteps` is the number of time steps and `features` is the number of features at each time step.

2. **Bidirectional LSTM Layers**

   - The model contains two Bidirectional LSTM layers, each with 64 units.
   - The Bidirectional wrapper allows each LSTM to process sequences in both forward and backward directions, capturing dependencies from both sides of the sequence.
   - `return_sequences=True` is used for the first two LSTM layers to ensure that the output at each time step is returned, which is necessary for stacking multiple LSTM layers.

3. **Fully Connected (Dense) Layer**

   - A dense layer with 12 units and a sigmoid activation function is used as the output layer.

Overfitting Measures

- Dropout layers are utilized after the LSTM and Dense layers to reduce the risk of overfitting by preventing the model from relying too heavily on any single feature or connection.

Batch Normalization

- Batch normalization is applied after each LSTM and Dense layer to normalize the activations, which helps in stabilizing and speeding up the training process by maintaining a consistent distribution of activations.

In [15]:
```
model = Sequential()

model.add(Bidirectional(LSTM(64, return_sequences=True), input_shape=(X_train.shape[1], X_train.shape[2])))
model.add(BatchNormalization())
model.add(Dropout(0.5))

model.add(Bidirectional(LSTM(64, return_sequences=False)))
model.add(BatchNormalization())
model.add(Dropout(0.2))

model.add(Dense(64, activation='relu'))
model.add(BatchNormalization())
model.add(Dropout(0.1))

model.add(Dense(12, activation='sigmoid'))
```

```
WARNING:tensorflow:From c:\Users\PDesktop\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\ba
ckend.py:873: The name tf.get_default_graph is deprecated. Please use tf.compat.v1.get_default_graph instead.
```

Compile Model

Loss function:
Use the *Binary Crossentropy* loss function because it is a `binary multi-class classification` problem.

Optimizer:
Exploring the *Adam* optimizer.

Metrics:
The *accuracy* metric is used to evaluate the model.

In [16]:
```
model.compile(loss='binary_crossentropy',optimizer='adam',metrics=['accuracy'])
```

```
WARNING:tensorflow:From c:\Users\PDesktop\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\op
timizers\__init__.py:309: The name tf.train.Optimizer is deprecated. Please use tf.compat.v1.train.Optimizer ins
tead.
```

In [17]:
```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #   
=================================================================
 bidirectional (Bidirection  (None, 1, 128)            39424     
 al)                                                             
                                                                 
 batch_normalization (Batch  (None, 1, 128)            512       
 Normalization)                                                  
                                                                 
 dropout (Dropout)           (None, 1, 128)            0         
                                                                 
 bidirectional_1 (Bidirecti  (None, 128)               98816     
 onal)                                                           
                                                                 
 batch_normalization_1 (Bat  (None, 128)               512       
 chNormalization)                                                
                                                                 
 dropout_1 (Dropout)         (None, 128)               0         
                                                                 
 dense (Dense)               (None, 64)                8256      
                                                                 
 batch_normalization_2 (Bat  (None, 64)                256       
 chNormalization)                                                
                                                                 
 dropout_2 (Dropout)         (None, 64)                0         
                                                                 
 dense_1 (Dense)             (None, 12)                780       
                                                                 
=================================================================
Total params: 148556 (580.30 KB)
Trainable params: 147916 (577.80 KB)
Non-trainable params: 640 (2.50 KB)
_________________________________________________________________
```

Train Model

- 30 epochs
- Batch size of 64
- Early stopping
- Model checkpoint

In [18]:

```python
# best callback for the model
best_model_file = 'runtime_saves/models/checkpoints/1A-BidirecionalLSTM_CP.keras'
best_model = ModelCheckpoint(best_model_file, monitor='val_loss', mode='min', verbose=1, save_best_only=True)

# early stopping callback
early_stop = EarlyStopping(monitor='val_loss', mode='min', verbose=1, patience=10)

# fit the model
history = model.fit(X_train, y_train, epochs=30, batch_size=64 , validation_data=(X_test, y_test), verbose=1, ca
```

```
Epoch 1/30
WARNING:tensorflow:From c:\Users\PDesktop\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\ut
ils\tf_utils.py:492: The name tf.ragged.RaggedTensorValue is deprecated. Please use tf.compat.v1.ragged.RaggedTe
nsorValue instead.

WARNING:tensorflow:From c:\Users\PDesktop\AppData\Local\Programs\Python\Python311\Lib\site-packages\keras\src\en
gine\base_layer_utils.py:384: The name tf.executing_eagerly_outside_functions is deprecated. Please use tf.compa
t.v1.executing_eagerly_outside_functions instead.

406/412 [============================>.] - ETA: 0s - loss: 0.2790 - accuracy: 0.2096
Epoch 1: val_loss improved from inf to 0.08025, saving model to runtime_saves/models/checkpoints\1A-Bidirecional
LSTM_CP.keras
412/412 [==============================] - 16s 13ms/step - loss: 0.2761 - accuracy: 0.2105 - val_loss: 0.0803 -
val_accuracy: 0.0856
Epoch 2/30
403/412 [============================>.] - ETA: 0s - loss: 0.0393 - accuracy: 0.2104
Epoch 2: val_loss improved from 0.08025 to 0.02098, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0392 - accuracy: 0.2102 - val_loss: 0.0210 - va
l_accuracy: 0.2648
Epoch 3/30
412/412 [==============================] - ETA: 0s - loss: 0.0287 - accuracy: 0.2028
Epoch 3: val_loss improved from 0.02098 to 0.01688, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 4s 10ms/step - loss: 0.0287 - accuracy: 0.2028 - val_loss: 0.0169 - v
al_accuracy: 0.1900
Epoch 4/30
410/412 [============================>.] - ETA: 0s - loss: 0.0243 - accuracy: 0.2007
Epoch 4: val_loss improved from 0.01688 to 0.01293, saving model to runtime_saves/models/checkpoints\1A-Bidireci
```

```
onalLSTM_CP.keras
412/412 [==============================] - 3s 8ms/step - loss: 0.0243 - accuracy: 0.2008 - val_loss: 0.0129 - va
l_accuracy: 0.2074
Epoch 5/30
405/412 [============================>.] - ETA: 0s - loss: 0.0221 - accuracy: 0.2012
Epoch 5: val_loss improved from 0.01293 to 0.01260, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 4s 10ms/step - loss: 0.0220 - accuracy: 0.2009 - val_loss: 0.0126 - v
al_accuracy: 0.2243
Epoch 6/30
406/412 [============================>.] - ETA: 0s - loss: 0.0205 - accuracy: 0.2113
Epoch 6: val_loss improved from 0.01260 to 0.01081, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 3s 8ms/step - loss: 0.0205 - accuracy: 0.2112 - val_loss: 0.0108 - va
l_accuracy: 0.2435
Epoch 7/30
412/412 [==============================] - ETA: 0s - loss: 0.0197 - accuracy: 0.2189
Epoch 7: val_loss improved from 0.01081 to 0.01059, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0197 - accuracy: 0.2189 - val_loss: 0.0106 - va
l_accuracy: 0.2566
Epoch 8/30
410/412 [============================>.] - ETA: 0s - loss: 0.0186 - accuracy: 0.2180
Epoch 8: val_loss improved from 0.01059 to 0.01020, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0186 - accuracy: 0.2179 - val_loss: 0.0102 - va
l_accuracy: 0.2129
Epoch 9/30
408/412 [============================>.] - ETA: 0s - loss: 0.0179 - accuracy: 0.2143
Epoch 9: val_loss improved from 0.01020 to 0.00996, saving model to runtime_saves/models/checkpoints\1A-Bidireci
onalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0179 - accuracy: 0.2142 - val_loss: 0.0100 - va
l_accuracy: 0.2245
Epoch 10/30
406/412 [============================>.] - ETA: 0s - loss: 0.0180 - accuracy: 0.2096
Epoch 10: val_loss improved from 0.00996 to 0.00940, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 3s 8ms/step - loss: 0.0180 - accuracy: 0.2095 - val_loss: 0.0094 - va
l_accuracy: 0.2230
Epoch 11/30
408/412 [============================>.] - ETA: 0s - loss: 0.0174 - accuracy: 0.2164
Epoch 11: val_loss improved from 0.00940 to 0.00937, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0174 - accuracy: 0.2163 - val_loss: 0.0094 - va
l_accuracy: 0.2562
Epoch 12/30
405/412 [============================>.] - ETA: 0s - loss: 0.0164 - accuracy: 0.2223
Epoch 12: val_loss improved from 0.00937 to 0.00856, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0164 - accuracy: 0.2221 - val_loss: 0.0086 - va
l_accuracy: 0.2654
Epoch 13/30
412/412 [==============================] - ETA: 0s - loss: 0.0162 - accuracy: 0.2170
Epoch 13: val_loss did not improve from 0.00856
412/412 [==============================] - 3s 6ms/step - loss: 0.0162 - accuracy: 0.2170 - val_loss: 0.0097 - va
l_accuracy: 0.2312
Epoch 14/30
409/412 [============================>.] - ETA: 0s - loss: 0.0163 - accuracy: 0.2234
Epoch 14: val_loss did not improve from 0.00856
412/412 [==============================] - 3s 6ms/step - loss: 0.0163 - accuracy: 0.2231 - val_loss: 0.0089 - va
l_accuracy: 0.2258
Epoch 15/30
405/412 [============================>.] - ETA: 0s - loss: 0.0156 - accuracy: 0.2203
Epoch 15: val_loss did not improve from 0.00856
412/412 [==============================] - 3s 6ms/step - loss: 0.0156 - accuracy: 0.2198 - val_loss: 0.0089 - va
l_accuracy: 0.1997
Epoch 16/30
411/412 [============================>.] - ETA: 0s - loss: 0.0150 - accuracy: 0.2097
Epoch 16: val_loss did not improve from 0.00856
412/412 [==============================] - 3s 8ms/step - loss: 0.0150 - accuracy: 0.2098 - val_loss: 0.0086 - va
l_accuracy: 0.2302
Epoch 17/30
411/412 [============================>.] - ETA: 0s - loss: 0.0151 - accuracy: 0.2103
Epoch 17: val_loss did not improve from 0.00856
412/412 [==============================] - 4s 11ms/step - loss: 0.0151 - accuracy: 0.2103 - val_loss: 0.0089 - v
al_accuracy: 0.2072
Epoch 18/30
405/412 [============================>.] - ETA: 0s - loss: 0.0147 - accuracy: 0.2091
Epoch 18: val_loss improved from 0.00856 to 0.00799, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 5s 11ms/step - loss: 0.0147 - accuracy: 0.2094 - val_loss: 0.0080 - v
al_accuracy: 0.2148
Epoch 19/30
```

```
410/412 [============================>.] - ETA: 0s - loss: 0.0142 - accuracy: 0.2139
Epoch 19: val_loss improved from 0.00799 to 0.00792, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 3s 8ms/step - loss: 0.0142 - accuracy: 0.2140 - val_loss: 0.0079 - va
l_accuracy: 0.2089
Epoch 20/30
405/412 [============================>.] - ETA: 0s - loss: 0.0135 - accuracy: 0.2227
Epoch 20: val_loss improved from 0.00792 to 0.00779, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 5s 12ms/step - loss: 0.0135 - accuracy: 0.2229 - val_loss: 0.0078 - v
al_accuracy: 0.2602
Epoch 21/30
409/412 [============================>.] - ETA: 0s - loss: 0.0136 - accuracy: 0.2289
Epoch 21: val_loss improved from 0.00779 to 0.00763, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 4s 9ms/step - loss: 0.0135 - accuracy: 0.2286 - val_loss: 0.0076 - va
l_accuracy: 0.2632
Epoch 22/30
412/412 [==============================] - ETA: 0s - loss: 0.0134 - accuracy: 0.2309
Epoch 22: val_loss improved from 0.00763 to 0.00677, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 4s 11ms/step - loss: 0.0134 - accuracy: 0.2309 - val_loss: 0.0068 - v
al_accuracy: 0.2725
Epoch 23/30
406/412 [============================>.] - ETA: 0s - loss: 0.0132 - accuracy: 0.2311
Epoch 23: val_loss did not improve from 0.00677
412/412 [==============================] - 5s 13ms/step - loss: 0.0133 - accuracy: 0.2313 - val_loss: 0.0071 - v
al_accuracy: 0.2693
Epoch 24/30
404/412 [============================>.] - ETA: 0s - loss: 0.0126 - accuracy: 0.2227
Epoch 24: val_loss did not improve from 0.00677
412/412 [==============================] - 4s 9ms/step - loss: 0.0127 - accuracy: 0.2231 - val_loss: 0.0079 - va
l_accuracy: 0.2118
Epoch 25/30
410/412 [============================>.] - ETA: 0s - loss: 0.0126 - accuracy: 0.2311
Epoch 25: val_loss did not improve from 0.00677
412/412 [==============================] - 3s 6ms/step - loss: 0.0126 - accuracy: 0.2310 - val_loss: 0.0074 - va
l_accuracy: 0.2693
Epoch 26/30
409/412 [============================>.] - ETA: 0s - loss: 0.0127 - accuracy: 0.2348
Epoch 26: val_loss did not improve from 0.00677
412/412 [==============================] - 3s 7ms/step - loss: 0.0126 - accuracy: 0.2348 - val_loss: 0.0077 - va
l_accuracy: 0.2621
Epoch 27/30
404/412 [============================>.] - ETA: 0s - loss: 0.0123 - accuracy: 0.2330
Epoch 27: val_loss did not improve from 0.00677
412/412 [==============================] - 3s 7ms/step - loss: 0.0123 - accuracy: 0.2328 - val_loss: 0.0084 - va
l_accuracy: 0.2102
Epoch 28/30
410/412 [============================>.] - ETA: 0s - loss: 0.0121 - accuracy: 0.2425
Epoch 28: val_loss improved from 0.00677 to 0.00635, saving model to runtime_saves/models/checkpoints\1A-Bidirec
ionalLSTM_CP.keras
412/412 [==============================] - 3s 7ms/step - loss: 0.0121 - accuracy: 0.2426 - val_loss: 0.0063 - va
l_accuracy: 0.2450
Epoch 29/30
411/412 [============================>.] - ETA: 0s - loss: 0.0117 - accuracy: 0.2446
Epoch 29: val_loss did not improve from 0.00635
412/412 [==============================] - 4s 9ms/step - loss: 0.0117 - accuracy: 0.2446 - val_loss: 0.0065 - va
l_accuracy: 0.3158
Epoch 30/30
408/412 [============================>.] - ETA: 0s - loss: 0.0117 - accuracy: 0.2426
Epoch 30: val_loss did not improve from 0.00635
412/412 [==============================] - 3s 6ms/step - loss: 0.0117 - accuracy: 0.2425 - val_loss: 0.0067 - va
l_accuracy: 0.2118
```

Save Model

In [19]:
```
model.save('runtime_saves/models/1A-BidirecionalLSTM.keras')
```

Load Model

In [13]:
```
from keras.models import load_model

model = load_model('runtime_saves/models/1A-BidirecionalLSTM.keras')

X_test = np.load('runtime_saves/data/X_test.npy')
y_test = np.loadtxt('runtime_saves/data/y_test.csv', delimiter=',', dtype=int)
```

RESULTS AND EVALUATION

Training history

In [ ]:
```
#plot loss e val_loss
plt.figure(figsize=(10, 6))
plt.plot(history.history['loss'], label='loss')
plt.plot(history.history['val_loss'], label='val_loss')
plt.xlabel('Epoch')
plt.ylabel('Loss')
plt.title('Loss vs. Epoch')
plt.legend()
plt.show()
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Cell In[87], line 3
      1 #plot loss e val_loss
      2 plt.figure(figsize=(10, 6))
----> 3 plt.plot(history.history['loss'], label='loss')
      4 plt.plot(history.history['val_loss'], label='val_loss')
      5 plt.xlabel('Epoch')

AttributeError: 'list' object has no attribute 'history'
<Figure size 1000x600 with 0 Axes>
```

Performance Metrics

- Accuracy $= \frac{Correct\ Predictions}{All\ Predictions}$ $= \frac{Correct\ Predictions}{All\ Predictions}$
- Precision for a given class $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Predictions\ for\ the\ Class}$ $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Predictions\ for\ the\ Class}$
- Recall for a given class $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Instances\ of\ the\ Class}$ $= \frac{Correct\ Predictions\ for\ the\ Class}{All\ Instances\ of\ the\ Class}$
- F1 Score $= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$ $= \frac{2 \times \text{Precision} \times \text{Recall}}{\text{Precision} + \text{Recall}}$
- Hamming Loss $= \frac{1}{N} \sum_{i=1}^{N} \frac{\text{Incorrect Labels}}{\text{Total Labels}}$
- Jaccard Score $= \frac{|Y_{pred} \cap Y_{true}|}{|Y_{pred} \cup Y_{true}|}$
- Averaging is a way to get a single number for multiclass. Depending on the importance one wants to give to minority classes:
  - Macro average: Compute the metric for each class, and returns the average without considering the proportion for each class in the dataset. For instance:

    Precision = $= \frac{P_{class1} + P_{class2} + \ ...\ + P_{classn}}{N}$

  - Weighted average: Compute the metric for each class, and returns the average considering the proportion (weighted) for each class in the dataset. For instance:

    Precision = $= \frac{N_1 * P_{class1} + N_2 * P_{class2} + \ ...\ + N_n * P_{classn}}{N}$

In [20]:
```
y_pred = model.predict(X_test)

y_pred = np.round(y_pred)

metrics = {
    'Metric': ['Accuracy', 'Precision', 'Recall', 'F1 Score', 'Hamming Loss', 'Jaccard Score'],
    'Value': [
        accuracy_score(y_test, y_pred) * 100,
        precision_score(y_test, y_pred, average="weighted") * 100,
        recall_score(y_test, y_pred, average="weighted") * 100,
        f1_score(y_test, y_pred, average="weighted") * 100,
        hamming_loss(y_test, y_pred) * 100,
        jaccard_score(y_test, y_pred, average="weighted") * 100,
    ]
```

```
}

metrics_df = pd.DataFrame(metrics)

# Percentage formatting
metrics_df['Value'] = metrics_df['Value'].map('{:.2f}%'.format)

metrics_df
```

```
275/275 [==============================] - 1s 5ms/step
```

Out[20]:

	Metric	Value
0	Accuracy	96.96%
1	Precision	94.26%
2	Recall	90.90%
3	F1 Score	92.14%
4	Hamming Loss	0.26%
5	Jaccard Score	85.80%

Per-class Classification Report

In [ ]:

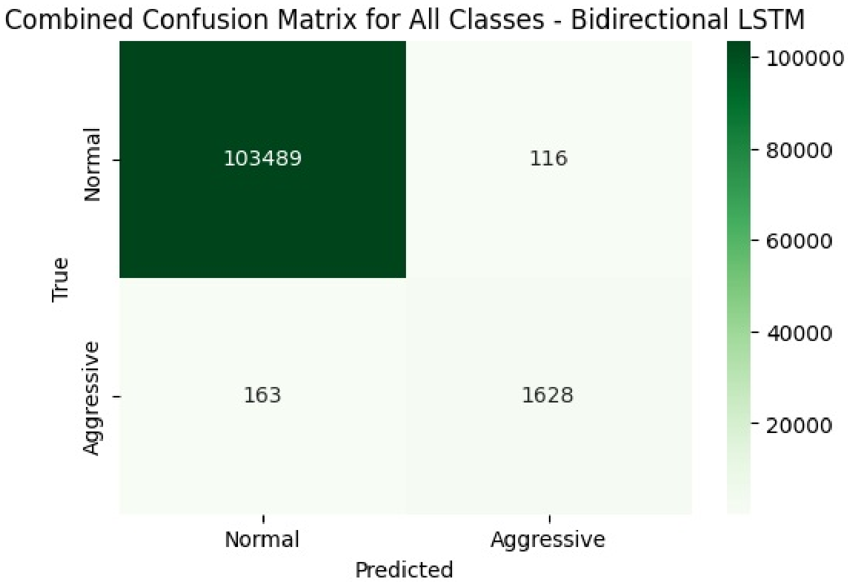
```
from sklearn.metrics import classification_report

# Print classification report for each class
class_report = classification_report(y_test, y_pred, target_names=maneuvers)
print(class_report)
```

```
                             precision    recall  f1-score   support

         Left Turn (Acc X+)       0.98      0.94      0.96       246
        Right Turn (Acc X-)       0.97      0.99      0.98       229
      Acceleration (Acc Y+)       0.76      0.96      0.85       190
           Braking (Acc Y-)       0.88      0.88      0.88        32
            Ascent (Acc Z+)       0.95      0.91      0.93        79
           Descent (Acc Z-)       0.69      0.86      0.77        29
 Left Lateral Tilt (Gyro X+)      0.99      0.87      0.92        99
Right Lateral Tilt (Gyro X-)      0.97      0.94      0.96       170
      Forward Tilt (Gyro Y+)      0.97      0.94      0.96       184
     Backward Tilt (Gyro Y-)      0.92      0.95      0.94       167
        Left Turn (Gyro Z+)       0.99      0.79      0.88       270
       Right Turn (Gyro Z-)       0.99      0.75      0.85        96

                  micro avg       0.93      0.91      0.92      1791
                  macro avg       0.92      0.90      0.91      1791
               weighted avg       0.94      0.91      0.92      1791
                samples avg       0.14      0.14      0.14      1791
```

Confusion Matrix

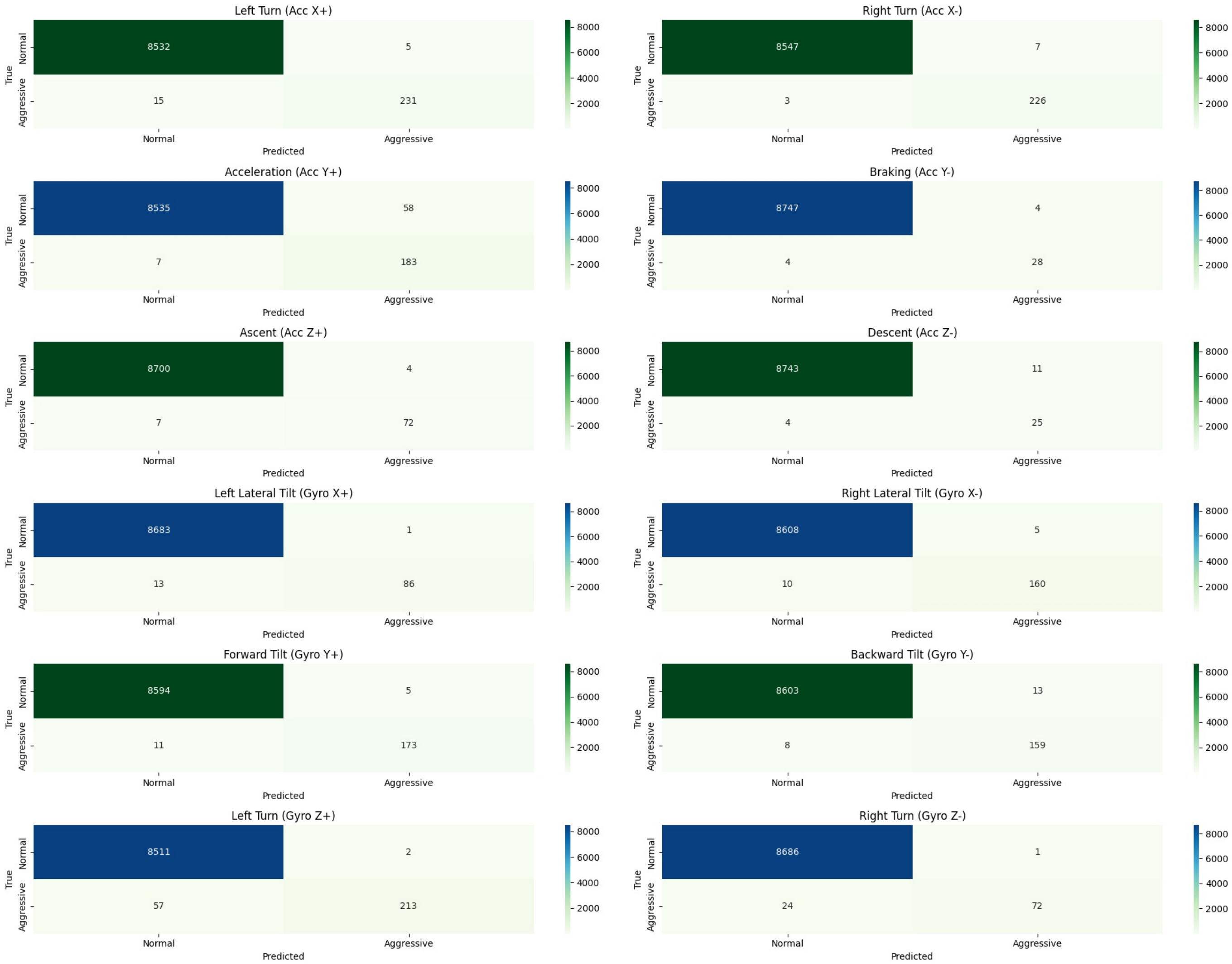
In [26]:

```
from sklearn.metrics import multilabel_confusion_matrix
# Binarize predictions at threshold 0.5
y_pred_classes = (y_pred > 0.5).astype(int)

# Multilabel confusion matrix
conf_matrices = multilabel_confusion_matrix(y_test, y_pred_classes)

# Aggregate confusion matrix
aggregated_conf_matrix = np.sum(conf_matrices, axis=0)

labels = ['Normal', 'Aggressive']

plt.figure(figsize=(6, 4))
sns.heatmap(aggregated_conf_matrix, annot=True, fmt="d", cmap="Greens",
            xticklabels=labels, yticklabels=labels)
plt.title('Combined Confusion Matrix for All Classes - Bidirectional LSTM')
plt.ylabel('True')
plt.xlabel('Predicted')
plt.show()
```



Per-Class Confusion Matrix

In [ ]:
```
# Plotting each confusion matrix for the 12 classes

green_color_maps = ['Greens', 'GnBu']

plt.figure(figsize=(20, 15))
for i, matrix in enumerate(conf_matrices):
    color_map = green_color_maps[(i // 2) % len(green_color_maps)]

    plt.subplot(6, 2, i + 1)
    sns.heatmap(matrix, annot=True, fmt="d", cmap=color_map,
                xticklabels=labels, yticklabels=labels)
    plt.title(maneuvers[i])
    plt.ylabel('True')
    plt.xlabel('Predicted')

plt.tight_layout()
plt.show()
```

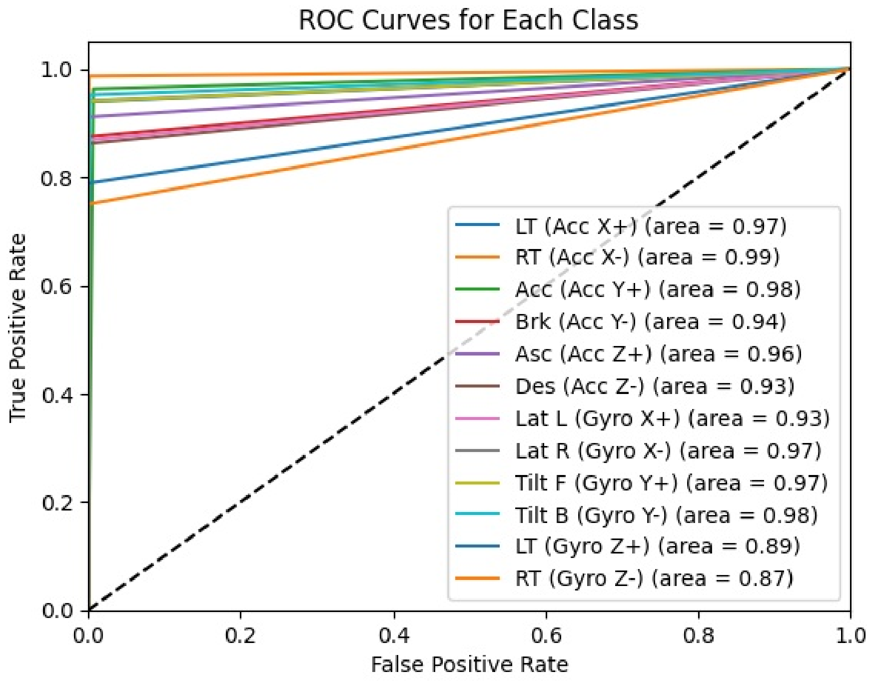


ROC Curve

The ROC curve visualizes a model's performance by plotting the True Positive Rate against the False Positive Rate at various thresholds. The Area Under the Curve (AUC) measures the model's ability to distinguish between classes, with a higher AUC indicating better performance.

In [25]:
```
from sklearn.metrics import roc_curve, auc

# Generate ROC Curve for each class
for i in range(12):
    fpr, tpr, _ = roc_curve(y_test[:, i], y_pred[:, i])
    roc_auc = auc(fpr, tpr)
    plt.plot(fpr, tpr, label=f'{maneuvers_short[i]} (area = {roc_auc:.2f})')

    # Print ROC AUC value
    print(f'{maneuvers[i]:<30} ROC AUC = {roc_auc:.2f}')

plt.plot([0, 1], [0, 1], 'k--')
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curves for Each Class - Bidirectional LSTM')
plt.legend(loc='lower right')
plt.show()
```

```
Left Turn (Acc X+)             ROC AUC = 0.97
Right Turn (Acc X-)            ROC AUC = 0.99
Acceleration (Acc Y+)          ROC AUC = 0.98
Braking (Acc Y-)               ROC AUC = 0.94
Ascent (Acc Z+)                ROC AUC = 0.96
Descent (Acc Z-)               ROC AUC = 0.93
Left Lateral Tilt (Gyro X+)    ROC AUC = 0.93
Right Lateral Tilt (Gyro X-)   ROC AUC = 0.97
Forward Tilt (Gyro Y+)         ROC AUC = 0.97
Backward Tilt (Gyro Y-)        ROC AUC = 0.98
Left Turn (Gyro Z+)            ROC AUC = 0.89
Right Turn (Gyro Z-)           ROC AUC = 0.87
```



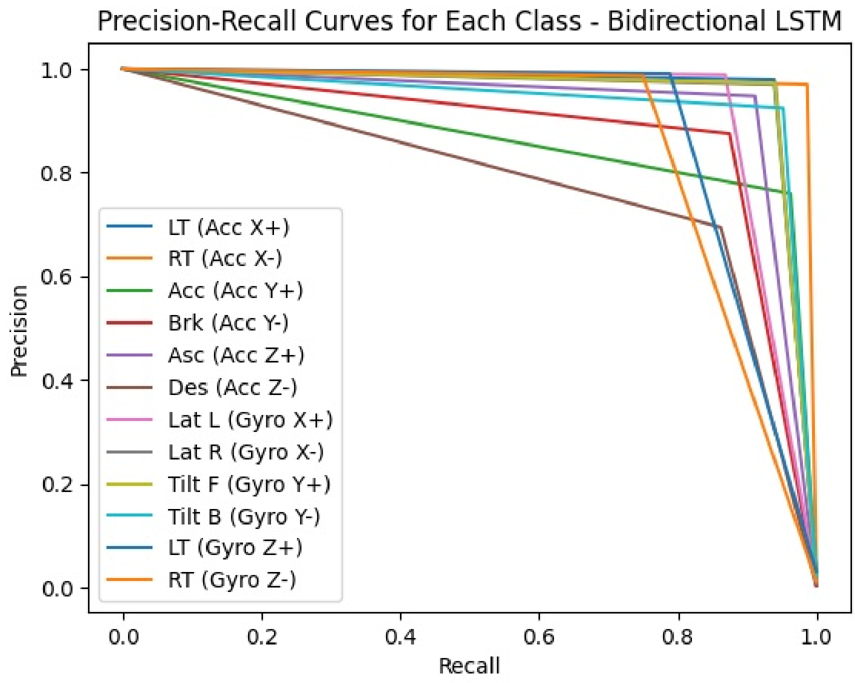
Precision-Recall curves

Precision-Recall curves illustrate a model's performance by plotting Precision against Recall for each class. These curves help evaluate how well the model balances precision and recall across different thresholds, especially in imbalanced datasets. Higher curves indicate better performance in identifying positive cases.

In [27]:

```
from sklearn.metrics import precision_recall_curve
import numpy as np
import matplotlib.pyplot as plt

plt.figure()

# Generate Precision-Recall Curve for each class
for i in range(12):
    precision, recall, _ = precision_recall_curve(y_test[:, i], y_pred[:, i])

    # Calculate maximum precision and recall
    max_precision_idx = precision.argmax()
    max_precision = precision[max_precision_idx]
    max_recall = recall[max_precision_idx]

    # Calculate precision at recall ~0.5
    threshold_idx = (np.abs(recall - 0.5)).argmin()
    precision_at = precision[threshold_idx] if threshold_idx < len(recall) else 'N/A'

    # Print summary for the class
    print(f'{maneuvers[i]:<30} Max Precision: {max_precision:.2f} at Recall: {max_recall:.2f}\t Precision at Re

    # Plot Precision-Recall Curve
    plt.plot(recall, precision, label=maneuvers_short[i])

# Plot settings
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.title('Precision-Recall Curves for Each Class - Bidirectional LSTM')
plt.legend(loc='lower left')
plt.show()
```

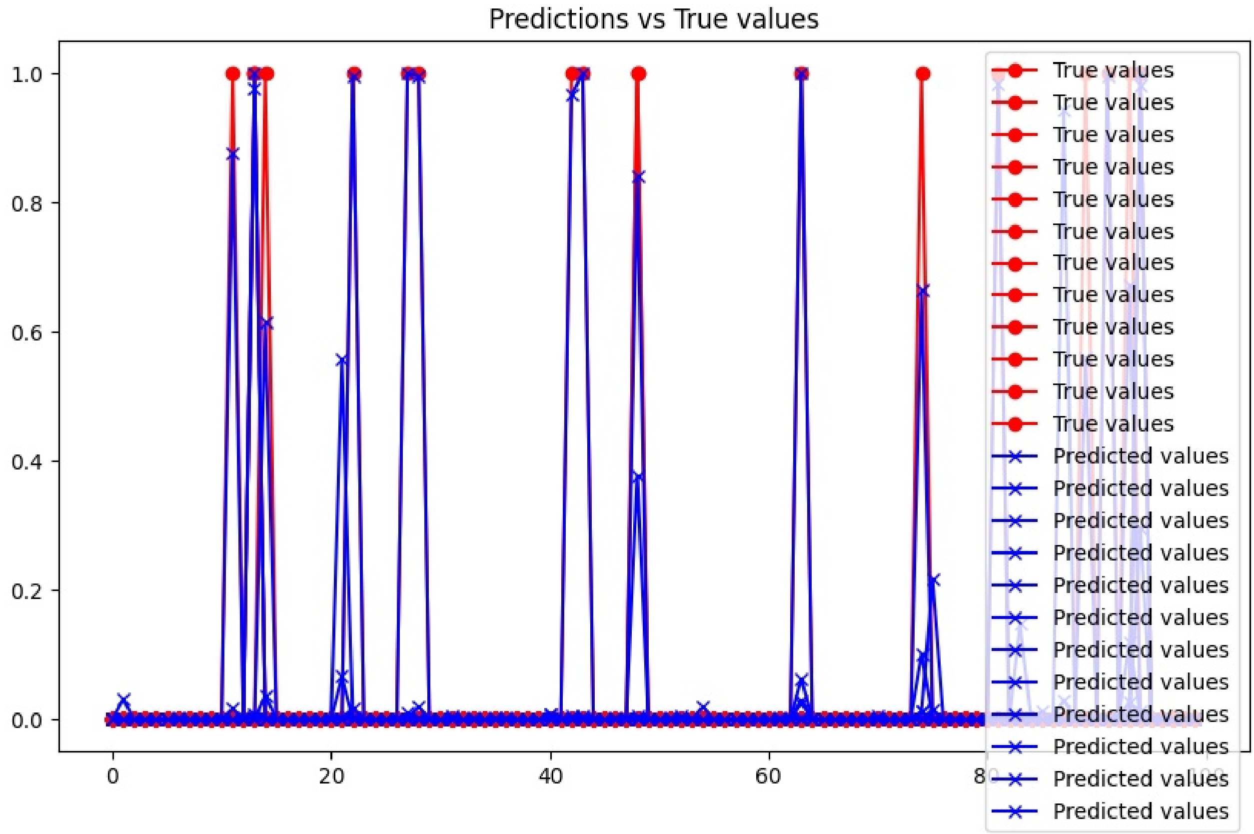
```
Left Turn (Acc X+)             Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.98
Right Turn (Acc X-)            Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.97
Acceleration (Acc Y+)          Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.76
Braking (Acc Y-)               Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.88
Ascent (Acc Z+)                Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.95
Descent (Acc Z-)               Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.69
Left Lateral Tilt (Gyro X+)    Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.99
Right Lateral Tilt (Gyro X-)   Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.97
Forward Tilt (Gyro Y+)         Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.97
Backward Tilt (Gyro Y-)        Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.92
Left Turn (Gyro Z+)            Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.99
Right Turn (Gyro Z-)           Max Precision: 1.00 at Recall: 0.00       Precision at Recall ~0.5: 0.99
```



Predict Test Data

In [ ]:
```python
def test_model(model, test_data, test_labels, start, end):
    predictions = model.predict(test_data[start:end])

    plt.figure(figsize=(10, 6))
    plt.plot(test_labels[start:end], 'ro-', label='True values')
    plt.plot(predictions, 'bx-', label='Predicted values')
    plt.title('Predictions vs True values')
    plt.legend()
    plt.show()

    return predictions

predictions = test_model(model, X_test, y_test, 0, 100)
```

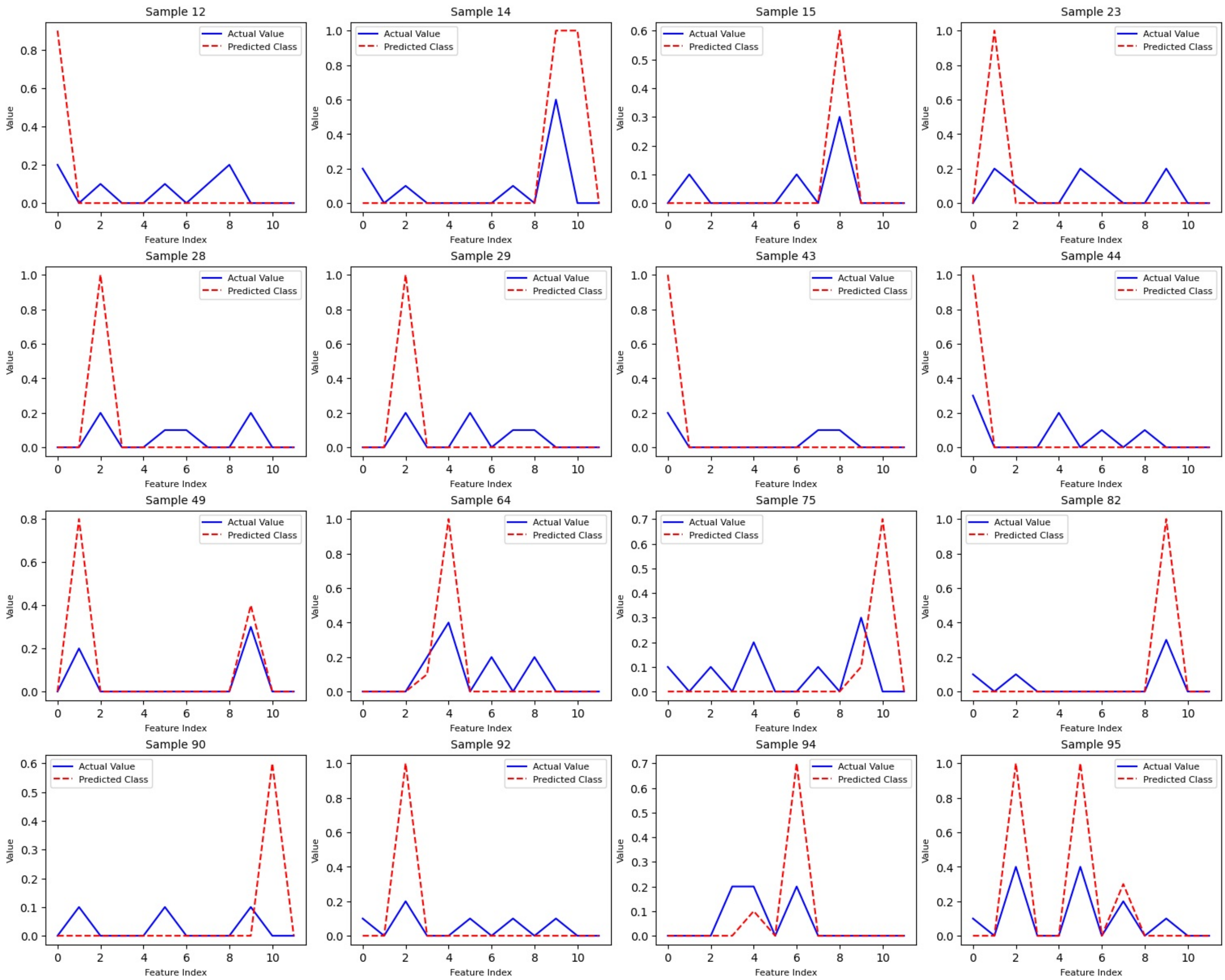
```
4/4 [==============================] - 0s 2ms/step
```



Visualize Predictions

In [ ]:
```
def visualize_predictions(x_test, y_test, model, num_samples=100):
    # List to store indices of valid samples
    valid_indices = []

    for i in range(num_samples):
        if np.sum(y_test[i]) > 0:
            valid_indices.append(i)

    num_valid_samples = len(valid_indices)

    if num_valid_samples == 0:
        print("No valid samples to plot.")
        return

    # Define the grid size based on the number of valid samples
    num_rows = int(math.ceil(num_valid_samples / 4))
    num_cols = min(num_valid_samples, 4)

    fig, axs = plt.subplots(num_rows, num_cols, figsize=(15, num_rows * 3), constrained_layout=True)

    axs = axs.flatten()

    for i, index in enumerate(valid_indices):
        a = x_test[index]
        b = a.reshape(1, 1, 12)

        prediction = model.predict(b)
        predicted_class = np.round(prediction.flatten(), decimals=1)

        actual_values = np.round(a.flatten(), decimals=1)
        true_class = np.round(y_test[index].flatten(), decimals=1)

        # Plot only if there is an actual class
        ax = axs[i]

        ax.plot(range(12), actual_values, 'b', label='Actual Value')
        ax.plot(range(12), predicted_class, 'r--', label='Predicted Class')

        ax.set_xlabel('Feature Index', fontsize=8)
        ax.set_ylabel('Value', fontsize=8)
        ax.set_title(f'Sample {index+1}', fontsize=10)
        ax.legend(fontsize=8)

    # Hide any remaining subplots that were not used
    for j in range(num_valid_samples, len(axs)):
        axs[j].axis('off')

    plt.show()

visualize_predictions(X_test, y_test, model, num_samples=100)
```

```
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 22ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 25ms/step
1/1 [==============================] - 0s 23ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 24ms/step
1/1 [==============================] - 0s 23ms/step
```



In [ ]:

```
import numpy as np
import matplotlib.pyplot as plt

test_value = np.array([0., 0.363, 0.313, 0., 0., 0.31, 0.393, 0., 0., 0.244, 0.247, 0.])
test_value = test_value.reshape(1, 1, 12)

prediction = model.predict(test_value)
prediction = prediction.flatten()
np.round(prediction, decimals=2, out=prediction)

print("Value    :", test_value[0][0])
print("Predicted:", prediction)

fig, ax = plt.subplots(figsize=(10, 6))

ax.plot(range(12), test_value[0][0], 'bo-', label='Test Value')

ax.plot(range(12), prediction, 'ro--', label='Predicted Value')

ax.set_xlabel('Feature Index', fontsize=14)
ax.set_ylabel('Value', fontsize=14)
ax.set_title('Test Value vs Predicted Value', fontsize=16)
ax.legend(fontsize=12)
plt.grid(True)
plt.tight_layout()
```

```
plt.show()
```

```
1/1 [==============================] - 0s 28ms/step
Value    : [0.    0.363 0.313 0.    0.    0.31  0.393 0.    0.    0.244 0.247 0.   ]
Predicted: [0.   0.94 0.88 0.1  0.01 0.   0.24 0.05 0.   0.18 1.   0.01]
```