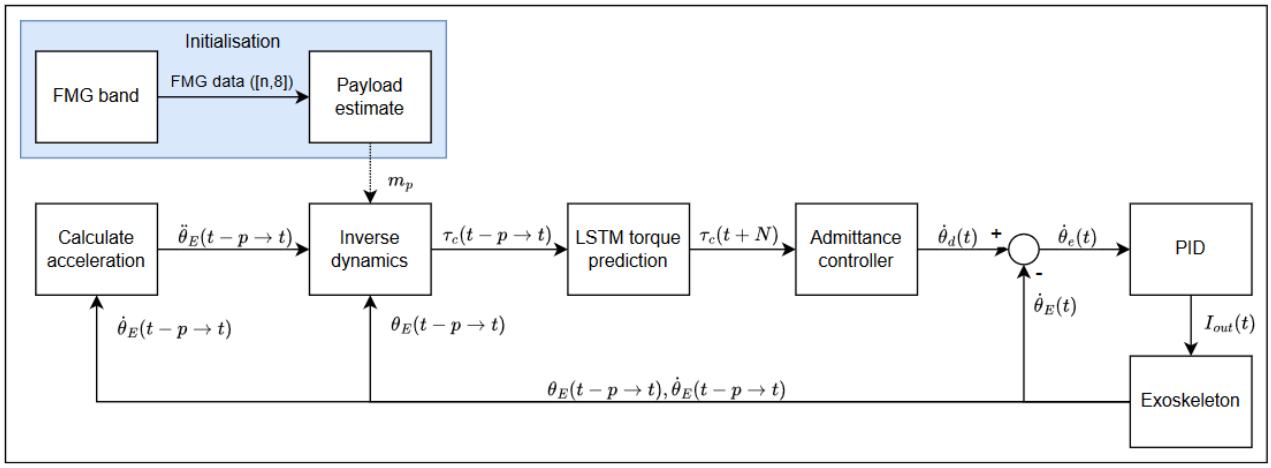
Documentation and Potential Improvements

Transparent control of 1-DOF exoskeleton using LSTM torque prediction and FMG based DNN weight classification.

Transparent control of exoskeleton system overview



Code Overview

Exoskeleton Control Code: Transparent-control-with-LSTM-and-FMG\1-Motor_Control\V1

- V_1.ino : Main loop, main motor controller, admittance control, PID Control, sensor data collection, implementation of Inverse Dynamics and LSTM torque prediction into admittance controller.
- InverseDynamics.hh : Class containing the LSTM torque predictor, the inverse dynamics model used to calculate torque from acceleration obtained from the encoder as well as scaling and differencing functions.
- model.h : The exported LSTM model. Predicts 1 time step into the future. constants_and_globals.h : source file containing constants and globals for the main code V_1.ino
- control_parameters.h : source file containing control parameters for V_1.ino. Change the parameters as needed.

LSTM Model Generation

- LSTM_ : Code that generates embedded LSTM model, see model.h file. Can be adjusted to generate model with different prediction horizons. Currently not working.

DNN Weight Classifier

- Weight Classifier : This code takes in 8xn matrices of FSR data and classifies the weight of the load held in the subjects hand. It distinguishes between 4 classes. Rest, flexed no load, flexed with 2.35kg load and flexed with 3.43kg load. This code is run offline and is not directly connected to the rest of the system. If connected to the system the classifier is meant to be an initializer that classifies the weight ONCE, and then does not run again until the exoskeleton is deliberately recalibrated. The classified weight is used in the Inverse Dynamics model to calculate the torque.
- FMG armband Code: This code runs the FMG band for 200 sec and collects an 8xn matrix of FSR data. This data can be saved and processed to be used in the Weight Classifier. To use the code connect the FMG band to the PC via Bluetooth. Identify the COM port of the Bluetooth device and

enter it into the CONNECT_AAL_BAND.m script and run it, if it is correctly identified it should not throw any errors. After that run the PID_clt_dq_singlebandtest000.m script and follow the instructions given in the terminal to collect data.

Collection of Torque data

- ReadDataFromSerialMonitor.m : This code collects data from an ESP32 serial monitor. It is used to collect data from the ESP32 that controls the exoskeleton motor. When running the V_1.ino motor control code on the Exoskeleton simply run this script in matlab and it will automatically read from the serial port connected to the ESP32. (Do not have Serial monitor open in Arduino IDE or it will not work).

Potential Improvements

- Fix the LSTM model generator ("Transparent-control-with-LSTM-and-FMG\LSTM Model generation\LSTM_.ipynb").
- If necessary, redo both the LSTM model generator and the LSTM model implementation on ESP32("Transparent-control-with-LSTM-and-FMG\1-Motor_Control\V1\InverseDynamics.hh").
- Reevaluate the Inverse Dynamics model used in the control code("Transparent-control-with-LSTM-and-FMG\1-Motor_Control\V1\InverseDynamics.hh"). See Exoskeleton research document for more details on the model.
- Redesign Weight Classifier so it can run online and classify weights based on raw data directly from the FMG band.
- Find a way to export the DNN weight classifier model to the ESP32 and connect the FMG band to the exoskeleton via Bluetooth. Alternatively, establish a Bluetooth connection from PC to ESP32 that sends the weight of the classified load from PC to ESP32.
- Switch the embedded LSTM model to a quantization aware model. (Currently uses post-training quantization)
- Try to incorporate the torque sensor into the control algorithm at the wrist to improve the responsiveness of the exoskeleton and reduce interaction torque. (1-Motor_Control/V_1/V1)
- Train a Reinforcement Learning model for torque prediction and compare transparency (Measured interaction Torque) against LSTM.

User Guide

Motor Control Code

Step 1: Specify payload mass and control status

```
35    float payloadMass = 1.1;
```
Line 35 V_1.ino

Payload mass should be set to the mass of the payload classified by the weight classifier. Or just whatever weight is being used for testing.

```
31    char control_status = 'D';//
```
Line 31 constants_and_globals.h

To enable the control algorithm then make control_status = ‘C’.

Step 2: Upload V_1.ino and dependencies to the ESP that controls the exoskeleton motors.

Step 3: Move the exoskeleton joint to confirm that the motor controller is working.

Important functions

- **InverseDynamics.Torque(): This function calculates the torque from acceleration and saves the 2 previously calculated torques.**

```
float Torque(float angle, float velocity, float mass,float sampleTime){
  mass2 = mass;
  prevVelocity = velocity;
  torque3 = torque2;
  torque2 = torque1;
  torque1 = (Je+mass*sq(lp))*acceleration;//+0*Me*le*g*sin(angle)+mass*lp*sin(angle)*g;
  return torque1;
}
```

- **InverseDynamics.LSTMpredict: This function performs the torque prediction using LSTM. It takes a differenced and scaled torque that has been passed through the diffTorque() and Scaling_transformation() functions as input.**

```
float LSTMpredict(float torque1scaled){
  float input[3] = {torque3scaled,torque2scaled,torque1scaled};

  while (!tf.predict(input).isOk())
      Serial.println(tf.exception.toString());

  tf.predict(input);
  torque3scaled = torque2scaled;
  torque2scaled = torque1scaled;
  return tf.result();
}
```

- **Calculating Acceleration: The acceleration is calculated in the V_1.ino code and saved to the InverseDynamics object in a float called acceleration.**

```
172   Exo_filter_data[1] = (lp_vals.ts * lp_vals.wc * Exo_filter_data[1] + lp_vals.old_vel_MR) / (1 + lp_vals.ts * lp_vals.wc);
173   right.acceleration = (Exo_filter_data[1]-lp_vals.old_vel_MR)/lp_vals.ts;
174   right.acceleration = (lp_vals.ts * lp_vals.wc * right.acceleration + right.prevAcceleration) / (1 + lp_vals.ts * lp_vals.wc);
175   lp_vals.old_vel_MR = Exo_filter_data[1];
176   right.prevAcceleration = right.acceleration;
```

The acceleration is calculated from velocity readings taken from the motor encoder. The values are lowpass filtered to remove noise.

DNN Weight Classifier

Step 1: Collect FSR data to classify

To collect data connect the FMG band to the PC via Bluetooth. Identify the COM port of the Bluetooth device and enter it into the CONNECT_AAL_BAND.m script and run it, if it is correctly identified it should not throw any errors. After that run the PID_clt_dq_singlebandtest000.m script and follow the instructions given in the terminal. While holding a weight of your choice perform flexion and extension of the arm until the script stops recording. Try to make the flexion and extension times be roughly the same lengths so the data is easier to process.

When the data is collected cut the data into chunks that separate the flexed state and the extended state. Move this data to an excel doc.

Repeat the process for different loads.

Alternatively, just use the data that was already given in the folder FMG band Data Collector/FMG Data

The docs labeled NoLoad = 0kg, Load = 2.35kg, LargeLoad = 3.43kg. Flexed = Flexion, Rested = Extension.

Step 2: Load the data into the Weight Classifier.ipynb script.

```
data = []
# "Rest" data
data.append( [pd.read_excel(constant_path + 'LouisFMGdata/LouisLoadRest/dataLouisLoadRest0001.xls',header=None), 0])
```

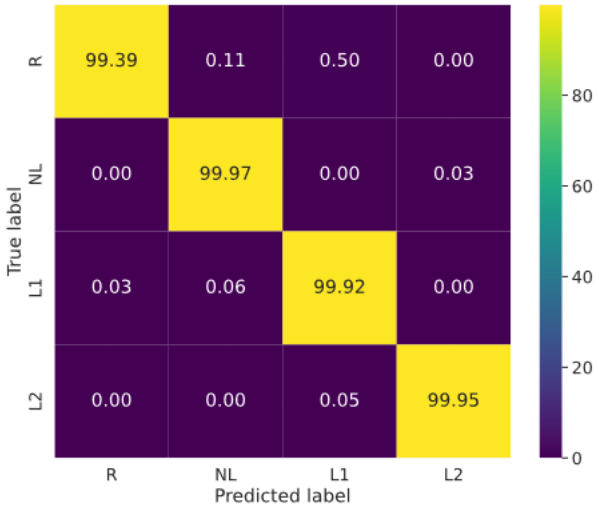
← replace the path with your own and specify the label for that set of data. In this case 0 means extension. The number of different labels you specify here will determine the number of classes your data will be classified into. The label should correspond to a specific payload in flexion or the rest state/extension.

The default configuration has 4 classes/labels, rest(0), flexed no load(1), flexed load 1(2) and flexed load 2(3)

Repeat for all data.append() lines until you have included all the data you feel you need.

Step 3: Run the script.

The script should give a confusion matrix as a result



LSTM Model Generation (Note: This code is not working)

Step 1: Record Torque data from the exoskeleton.

Run the Motor control code with the control disabled control status = 'D'.

Move the exoskeleton arm manually while running the matlab script ReadDataFromSerialMonitor.m

The matlab script makes a data matrix with 8 columns. The first column contains the torque data.

Save the collected data to an excel file.

Step 2: Load the collected data into the LSTM_.ipynb script

```
# load dataset
series = read_excel("/content/drive/MyDrive/LSTM/Inverse Dynamics Calculations/TorqueData_noFriction_NL_NG.xlsx", header=None, index_col=0)
```

Step 4: Specify the parameters of your desired model.

```
n_lag = 3
n_seq = 1
n_test = round(0.3*len(series))  # Or whatever number of tests you want
n_epochs = 10 # Number of epochs for training
n_batch = 1  # Batch size
n_neurons = 3  # Number of neurons in the LSTM layer
```

Step 4: Run the script

The script will give you a model that can be transferred to a model.h file that can be used in the control code.

The default setup will produce a model that will predict 1 time step in the future based on 3 previous torque readings