Modelado y simulación del Robot Mitsubishi RV-2JA controlado mediante señales electromiográficas

Busso, Francisco Ignacio
Gautero, Francisco
Gregoret, Guillermo

Universidad Tecnológica Nacional
Facultad Regional Santa Fe

5 de octubre de 2021

Objetivo

Controlar al Robot Mitsubishi RV-2JA mediante señales electromiográficas de superficie sEMG.

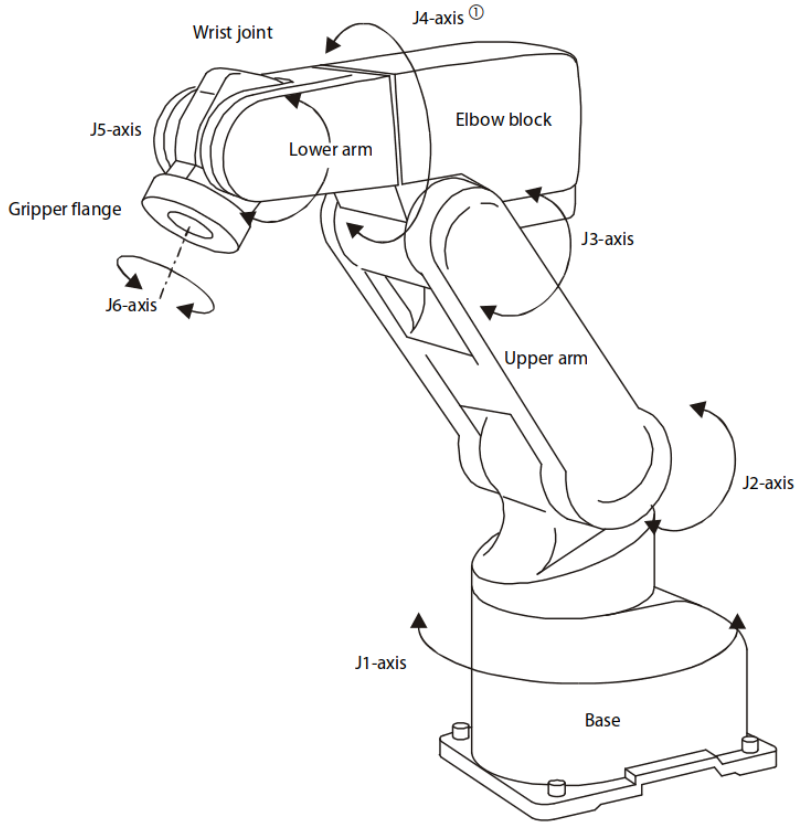


Figura: Robot Mitsubishi RV-2JA

Etapas del Trabajo

El trabajo original se divide en 3 aspectos principales:

1. Lectura y análisis de señales.
2. Reconocimiento de patrones a través de RNA.
3. Interfaz electrónica simulada.

Funcionamiento del Robot

El sistema electromiográfico está diseñado para que el operador controle al robot de forma remota; las señales sEMG son tomadas del operador y pertenecen a movimientos musculares específicos, estas son procesadas en el programa encargado de extraer sus características particulares para determinar el tipo de movimiento que realizará el robot.

Las características identificadas de las señales sEMG ingresan a una red neuronal artificial que activa las correspondientes articulaciones del robot que se desee mover con determinada posición de la mano.

Obtención de los Datos de Entrada

En el desarrollo del paper se utilizó el brazalete Myo como dispositivo de entrada para la adquisición de datos. Los datos de entrada son analizados mediante el diseño de un programa en Matlab-Simulink que proporciona datos de entrenamiento para las redes neuronales del sistema.

En un entorno virtual se realiza una simulación detallada del robot la cual permitirá replicar los movimientos deseados por el operador para luego validar los resultados del procesamiento y entrenamiento del sistema.

Modelo Simulink

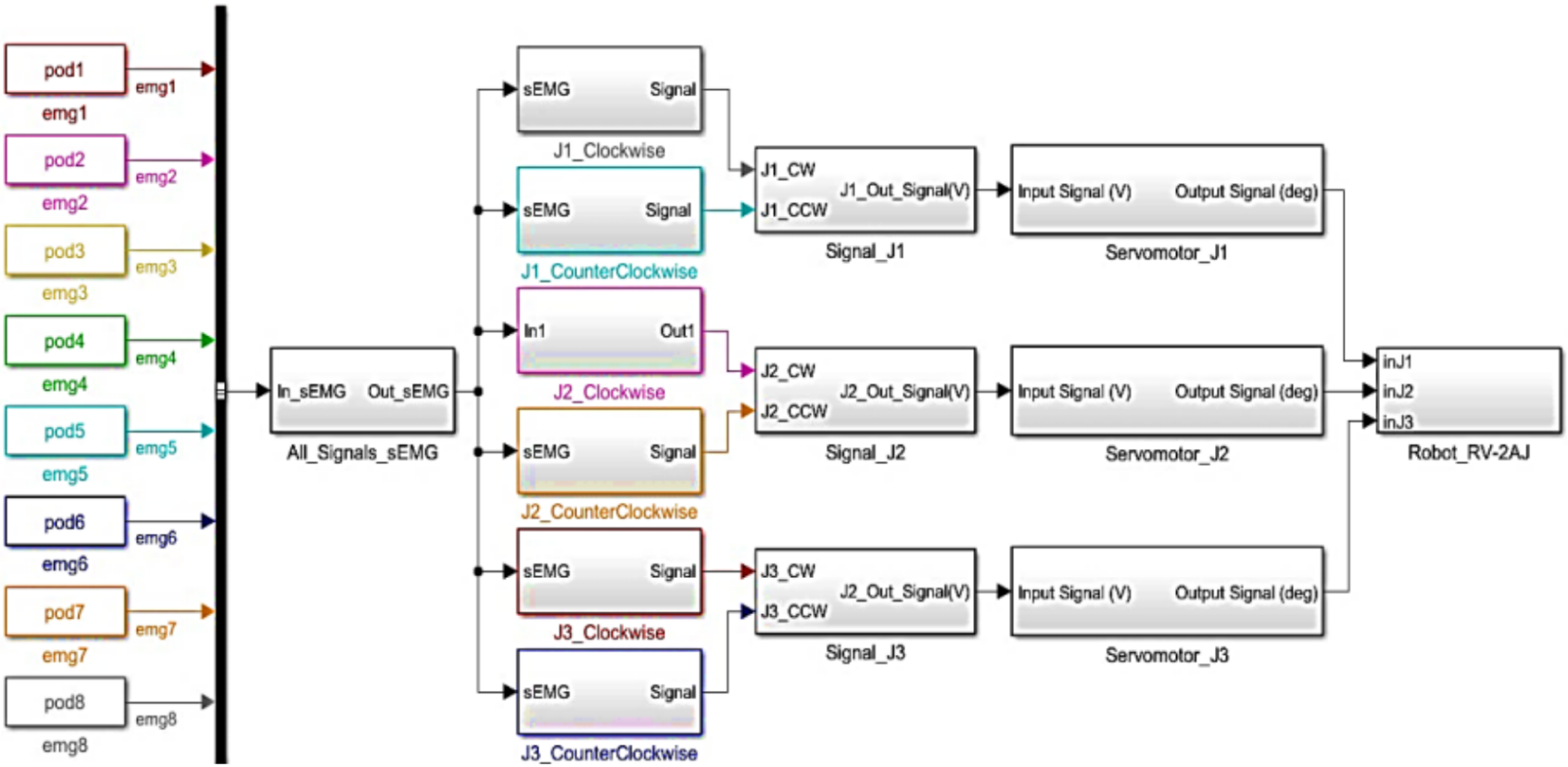


Figura: Arquitectura del modelo diseñado en Simulink.

Arquitectura de la Red Neuronal

Para la arquitectura de la red es necesario escoger el número de neuronas ocultas; con los parámetros seleccionados se procede al entrenamiento de la red usando un algoritmo de aprendizaje supervisado que se usa para entrenar redes neuronales artificiales. Se aplica el modelo Backpropagation que es la propagación hacia atrás de errores o retropropagación.

Cuando la red neuronal artificial ya está entrenada se generan versiones desplegables de la red neural entrenada.

Arquitectura de la Red Neuronal

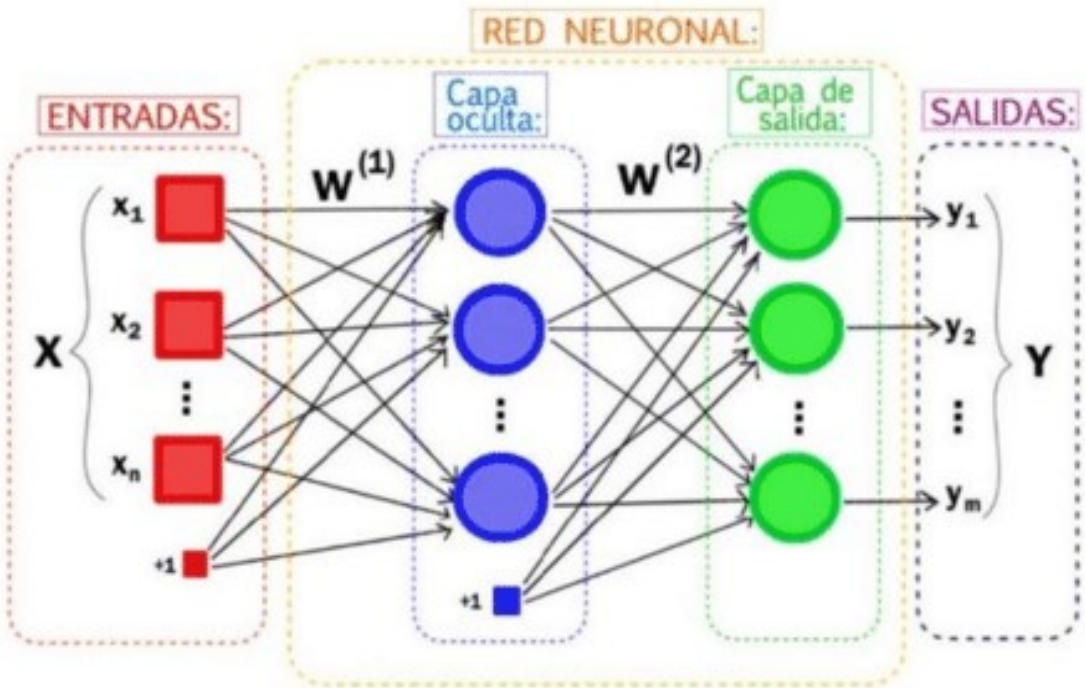


Figura: Arquitectura de la Red Neuronal

Dificultades encontradas

1. Resolución del análisis de señales.
2. Obtención de un dataset.

Nuestra Solución

Optamos por resolver una RNA capaz de detectar la posición FIST, ya que esta es la única que ejemplifica sus datos en el paper.

Tomamos los valores medios de cada lectura del sensor como valor original, y en base a ellos generamos un dataset propio definiendo un valor arbitrario de tolerancia.

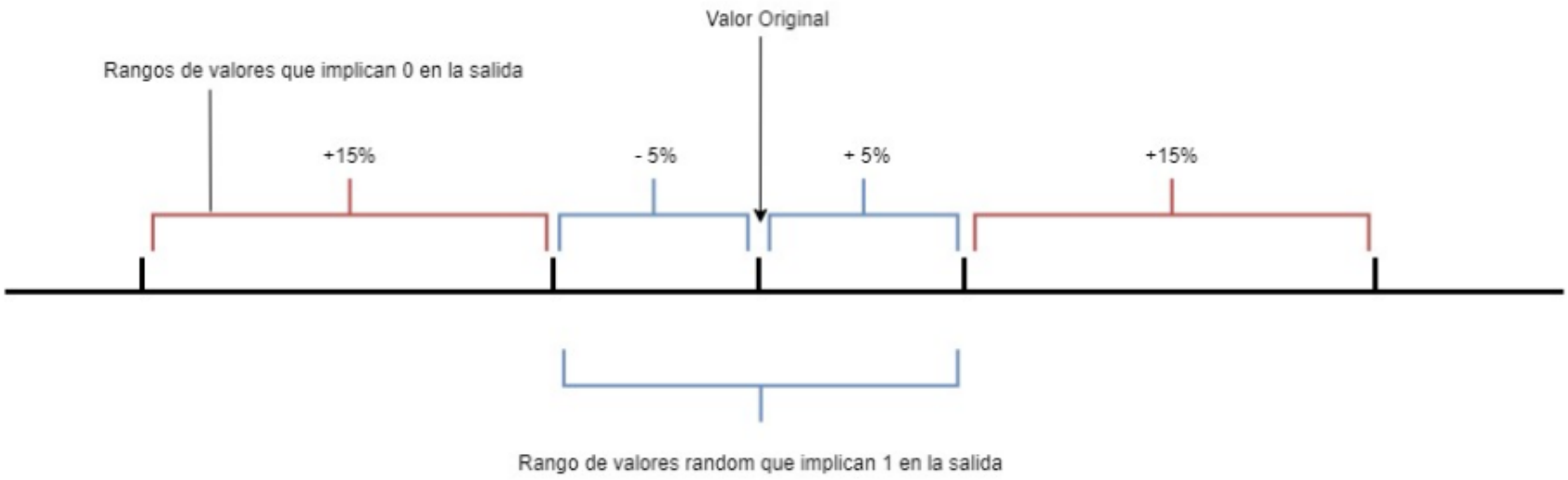
Valores de Referencia

	emg1	**emg2**	**em3**	**emg4**	**emg5**	**emg6**	**emg7**	**emg8**
Valor Medio	0.0449	0.0385	0.0481	0.0628	0.0897	0.0982	0.0814	0.0833
IEMG	0.5731	0.3816	0.1019	0.1996	0.7386	0.8177	1.1194	1.2671
MAV	0.0179	0.0119	0.0032	0.0062	0.0231	0.0256	0.0350	0.0396
RMS	0.0048	0.0035	0.0014	0.0026	0.0072	0.0082	0.0082	0.0097
VAR	0.0031	0.0014	0.0001	0.0006	0.0036	0.0036	0.0094	0.0116

Cuadro: Características de la señal sEMG, posición de mano FIST.

Valores de Referencia

Para generar valores de cada señal EMG se optó por una distribución uniforme con variación del 5 % sobre las señales de referencia.



Valores de Referencia

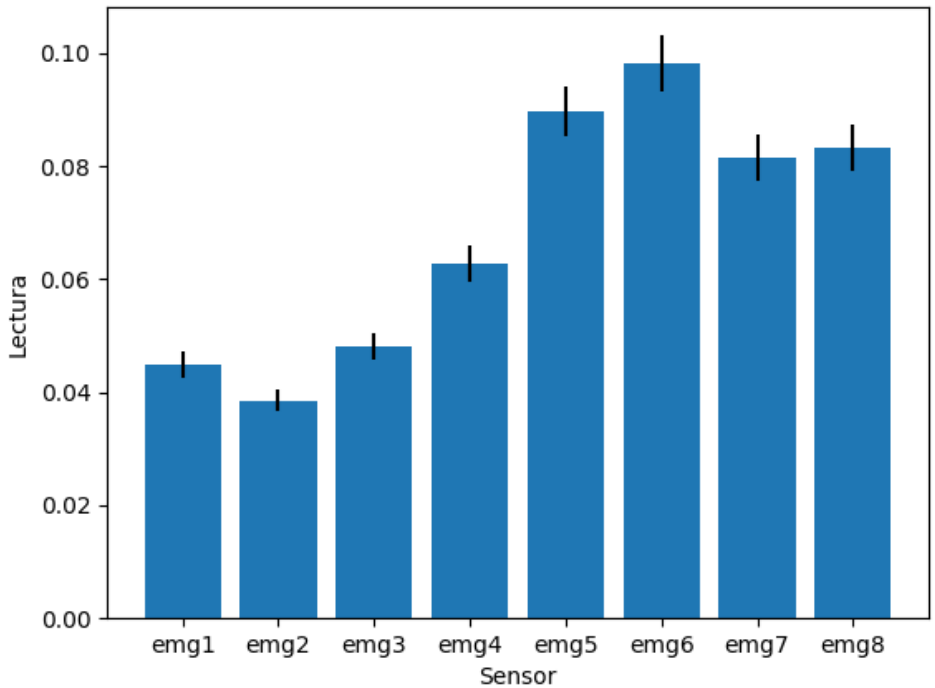


Figura: Valores de Referencia para cada sensor

Etiquetado del Set de Datos

Se define un valor umbral para determinar si el valor de la lectura de un sensor pertenece a los valores admitidos.

```
def in_range(value, index):
    x = [0.0449, 0.0385, 0.0481, 0.0628, 0.0897, 0.0982, 0.0814, 0.0833]
    return x[index] * 0.95 <= value <= x[index] * 1.05
```

Si las lecturas de todos los sensores se encuentran dentro del rango admitido, la entrada se etiqueta como 1; si no, se etiqueta como 0.

```
def tag(row):
    return 1.00 if all(in_range(row[i], i) for i in range(0, row.shape[0])) else 0.00
```

Obtención del Set de Entrenamiento

Se lee un set de datos, se lo etiqueta y se lo separa entre un set de entrenamiento y un set de pruebas.

```
>>> import pandas as pd
>>> import numpy as np
>>> from functions import normalize
>>> from sklearn.model_selection import train_test_split
>>> features = pd.read_csv('training_set.csv').to_numpy()
>>> labels = np.array(list(map(lambda x: normalize(x), features)))
>>> X_train, X_test, Y_train, Y_test = train_test_split(features,
                                                        labels,
                                                        test_size=0.50)
```

Implementación con scikit-learn

La clase MLPClassifier implementa un algoritmo de entrenamiento de perceptrón multicapa *(MLP)* utilizando propagación hacia atrás *(Backpropagation)*.

```
>>> from sklearn.neural_network import MLPClassifier
>>> clf = MLPClassifier(solver='lbfgs', alpha=1e-5,
                        hidden_layer_sizes=(20, 10),
                        random_state=1)
```

El Algoritmo MLP

El algoritmo toma como entrada dos arreglos: X, de dimensiones ($n - muestras$, $n - caracteristicas$), que representa las muestras de entrenamiento como vectores de punto flotante; e Y de tamaño ($n - muestras$), que almacena los valores de las etiquetas de clase para las muestras de entrenamiento.

```
>>> clf.fit(X_train, Y_train)
MLPClassifier(alpha=1e-05, hidden_layer_sizes=(20, 10),
              random_state=1,
              solver='lbfgs')
```

Estrategia de Resolución: Algoritmo L-BFGS

Es un algoritmo de optimización de la familia de los métodos *quasi-Newton* que aproxima al algoritmo (*BFGS*) utilizando una cantidad limitada de memoria. Es un algoritmo muy popular de estimación de parámetros en Machine Learning.

Para cada iteración el algoritmo busca una aproximación de la inversa de la matriz Hessiana, almacenando solo unos pocos vectores que representan a la aproximación implícitamente

Función de activación: Rectificador

En el contexto de las redes neuronales artificiales, el rectificador es una función de activación definida como:

$$ReLU(x) = x^{+} = max(0, x) \tag{1}$$

Donde x es la entrada de la neurona. También es conocida como función rampa y es análoga a la rectificación de media onda en electrónica.

Predicciones

Luego del ajuste (entrenamiento), el modelo puede predecir las etiquetas para nuevas muestras

```
>>> clf.predict(Y_test)
array([1.0, 0.0, 1.0, 0.0, 0.0, 1.0, 1.0])
```

Cálculo de Error

Se calculó el error comparando la salida esperada con la salida obtenida por el clasificador.

Neuronas Ocultas	**Error**
5	52.2 %
10	24.1 %
15	28.0 %
20	0.0 %

Cuadro: Numero de neuronas y error obtenido. Función de activación Rectificador.

Complejidad de la Solución

Suponiendo n muestras de entrenamiento, m características, k capas ocultas cada una conteniendo h neuronas y, para simplificar, o neuronas en la capa de salida. La complejidad temporal de la propagación hacia atrás es $O(n * m * h^k * o * i)$, donde i es el número de iteraciones.

Dado que el método de Backpropagation tiene alta complejidad temporal, es recomendable comenzar con un número bajo de neuronas en la capa oculta para el entrenamiento.

Conclusiones

Mediante el uso de una RNA MLP, entrenada con un algoritmo de backpropagation y datos creados artificialmente, se pudo identificar el patrón que refiere a la posición de maño fist de manera que el resultado es estable y repetible.

Conclusiones

Haciendo una comparación con los resultados originales en cuestión de error, no tuvimos como identificar la métrica observada ni como fue calculada. No obstante propusimos una métrica adecuada y podemos ver una relación similar error/cantidad

Cantidad de Neuronas Ocultas	**Error Obtenido en el Paper**	**Error Calculado**
5	4.39	52.2 %
10	4.29	24.1 %
15	4.44	28.0 %
20	4.29	0.0 %

Cuadro: Comparación del Error.

Conclusiones: Escalabilidad de la Solución

Esta implementación no está pensada para aplicaciones de gran escala. En particular, scikit-learn no ofrece soporte para GPU.

Código Fuente

El código fuente puede ser consultado en *GitHub*