PONTIFICIA UNIVERSIDAD CATÓLICA DEL PERÚ

FACULTAD DE CIENCIAS E INGENIERÍA

DISEÑO E IMPLEMENTACIÓN DE LAS FUNCIONES DE AGARRE Y LEVANTE EN UN BRAZO KINOVA USANDO SEÑALES EEG Y DEEP LEARNING

Tesis para optar el Título de Ingeniero Mecatrónico que presenta el bachiller:

Juan Manuel Neyra Pérez

ASESORA:

Elizabeth Roxana Villota Cerna, PhD

Lima, Julio, 2020

RESUMEN

Miles de personas en el mundo son afectadas por enfermedades causantes de parálisis tales como esclerosis lateral amiotrófica, lesiones en la médula espinal y distrofia muscular. En los últimos años, investigadores han buscado desarrollar soluciones tecnológicas para asistir a estos pacientes. En el 2012, una mujer con tetraplejia, causada por un paro cerebral, fue capaz de acercar una botella a su boca y beber de ella, utilizando señales EEG invasivas [1]. Recientemente, en el 2016, ahora mediante sensores EEG no invasivos, se realizaron pruebas en 13 sujetos sanos para mover un brazo robot en dos dimensiones [2]. Buscando colaborar en el desarrollo de robots asistenciales, el presente trabajo propone el diseño e implementación de las funciones de 'agarre' y 'levante' en el brazo robot Kinova, donde las señales de activación provendrán de señales EEG y el algoritmo de traducción estará basados en modelos de *deep learning*.

Los modelos de *deep learning* mencionados serán basados en la solución propuesta por Alex Barachant y Rafael Cycon para la clasificación de señales EEG [3]. El *dataset* que se utilizará para el entrenamiento se toma del repositorio WAY-EEG-GAL financiado por la unión europea [4]. A pesar de que las señales EEG corresponden a movimientos físicos reales, los cuales no pueden ser realizados por los pacientes con las enfermedades antes mencionadas, este trabajo busca brindar un aporte a la literatura médica e ingenieril y al avance de las aplicaciones de interfaz cerebro-computador. Adicionalmente, se busca proponer el método para evaluar el desempeño en una prueba experimental del algoritmo referido, lo cual no se ha abordado en la literatura presente hasta el momento.

Agradezco a mi familia por todo el apoyo que me han brindado y por estar conmigo durante todo el proceso. A mis padres y hermana, quienes han sido un gran soporte en la construcción de mi vida profesional. A mi asesora, la Dra. Elizabeth Roxana Villota Cerna, por su guía, por su ayuda para resolver mis dudas y brindarme las herramientas necesarias que permitieron el desarrollo de este proyecto. A mis amigos, de quienes aprendí cosas valiosas, que aportaron a mi desarrollo profesional.

ÍNDICE

Lista de Figuras

Lista de Tablas

Capítulo 1: INTRODUCCIÓN

En este capítulo, se discutirá la problemática; también se presentarán soluciones tecnológicas actuales, los objetivos, tanto el general como los específicos y los alcances del presente trabajo.

1.1. Discapacidad de destreza por edad avanzada y/o enfermedades crónicas

En el año 2012, se realizó la primera encuesta sobre discapacidad en el Perú. En ella, se muestra que el 59.2% del total de las personas con discapacidad presenta algún tipo de limitación en el uso de los brazos y/o piernas [5]. Además, de este total, el 82.9% sufre esta limitación con una severidad moderada o mayor. En las Figura 1.1 y Figura 1.2, se muestra el origen de las limitaciones, siendo más comunes la edad avanzada (32.7%), las enfermedades crónicas (25.1%) y los problemas congénitos (9.7%) [5]. Entre las causas de dificultad para el movimiento de brazos y piernas en personas de edad avanzada se encuentran la artritis, la osteoporosis y el Parkinson. Luego siguen las enfermedades crónicas entre las que se encuentran las lesiones en la médula espinal, la esclerosis lateral amiotrófica (ALS) y los infartos cerebrales. Finalmente, entre los problemas congénitos se encuentran la distrofia muscula y la ataxia de Friedreich. Dependiendo de la severidad de la situación, las personas con limitaciones de destreza no podrán desarrollar o desarrollarán con bastante dificultad actividades del día a día, como por ejemplo, hacer compras, vestirse, ducharse o alimentarse.

CUADRO Nº 6.12
PERÚ: PERSONAS CON DISCAPACIDAD PARA MOVERSE O CAMINAR, USAR BRAZOS Y MANOS / PIERNAS Y PIES POR ÁREA DE RESIDENCIA, SEGÚN ORIGEN DE LA LIMITACIÓN, 2012
(Porcentaje)

Origen de la limitación	Total	Urbana	Rural
Genético/congénito/de nacimiento	9,7	8,7	13,5
Enfermedad común	6,8	7,0	6,0
Enfermedad crónica	25,1	27,6	15,8
Enfermedad laboral	1,2	1,3	0,9
Accidente común en el hogar	4,6	4,5	5,3
Accidente común fuera del hogar	4,9	3,7	9,7
Accidente de tránsito	4,1	4,2	3,7
Accidente laboral	2,7	2,5	3,4
Actividades deportivas y recreativas	0,4	0,3	0,4
Violencia común	0,4	0,3	0,7
Violencia familiar	0,4	0,3	0,6

Continúa...

Figura 1.1. Porcentaje de personas con limitaciones de locomoción y destreza según origen de la limitación (parte 1). Fuente: [5]

CUADRO Nº 6.12
PERÚ: PERSONAS CON DISCAPACIDAD PARA MOVERSE O CAMINAR, USAR BRAZOS Y MANOS / PIERNAS Y PIES POR ÁREA DE RESIDENCIA, SEGÚN ORIGEN DE LA LIMITACIÓN, 2012
(Porcentaje)

Conclusión.

Origen de la limitación	Total	Urbana	Rural
Violencia sociopolítica	0,3	0,1	0,7
Fenómeno natural	0,9	0,7	1,6
Edad avanzada	32,7	32,9	31,9
Negligencia médica	1,4	1,3	1,8
Falta de atención médica	0,8	0,7	1,1
No buscó atención médica	0,5	0,5	0,5
Efectos colaterales de medicamentos	0,2	0,1	0,2
Alcohol, tabaco y otras drogas	0,1	0,1	0,1
Otro	3,6	4,2	1,1
No conoce el origen	3,3	3,1	3,9

Nota: Preguntas con respuestas múltiples.
Nota técnica: Los porcentajes han sido calculados respecto del total de personas con discapacidad para moverse o caminar.
Fuente: Instituto Nacional de Estadística e Informática - Primera Encuesta Nacional Especializada Sobre Discapacidad 2012.

Figura 1.2. Porcentaje de personas con limitaciones de locomoción y destreza según origen de la limitación (parte 2). Fuente: [5]

Estas dificultades físicas son también acompañadas de problemas emocionales, tanto para el paciente como para sus familiares. Paul R. Radcliff solía tener bastante fuerza en sus brazos y manos; sin embargo, en febrero del 2012, Paul le contó a su esposa que está perdiendo la fuerza en sus manos, no podía levantar ciertas cosas que antes podía, ni tampoco abrir una lata de cola [6]. Aunque primero ambos pensaron que la causa era el estrés, su pérdida muscular empeoró muy rápidamente, así que decidieron consultar con un doctor. El diagnóstico médico confirmó que Paul sufría de ALS y su condición empezó a deteriorase hasta llegar al punto de parálisis completa. Sin embargo, como se verá en la siguiente sección, existen soluciones en progreso que pueden aliviar estas dificultades.

1.2. Tecnología de asistencia para limitaciones de destreza

La tecnología juega un rol importante en la vida de personas con discapacidad de destreza. Jan Scheuermann, una mujer de 53 años, fue diagnosticada con degeneración espino cerebral en 1998 [7]. Actualmente, ella ya no puede mover sus brazos, o incluso levantar sus hombros. Sin embargo, gracias al esfuerzo de un equipo de investigación de la Universidad de Pittsburgh, Jan ha sido capaz de utilizar un brazo robot, cuyo efector final puede mover en siete dimensiones (3 orientación, 3 posición y 1 agarre) a través de implantes de microelectrodos en su córtex motor izquierdo. En la Figura 1.3, se observa como Jan comanda el brazo para acercar y comer una barra de chocolate. Como ella anunció ese día: “Este es un pequeño mordisco para una mujer, pero un gran bocado para las interfaces cerebro-computadora (BCI)”.

Actualmente, la tecnología de electroencefalograma (EEG) ha cobrado mayor importancia que en años previos, esto debido al surgimiento de técnicas no invasivas para la detección de señales EEG. Asimismo, las señales EEG tienen un rol muy importante cuando el paciente sufre de desórdenes del movimiento o parálisis, esto es debido a que con ellas se pueden

implementar dos tipos de soluciones de asistencia. Una de ellas es emplear exoesqueletos, utilizarlos con el fin de corregir y asistir en el movimiento del paciente, o como parte de un proceso de rehabilitación para él o ella [8]. La segunda es disponer de un brazo robot, que a través de una interfaz cerebro computadora (BCI) y un amplificador de señales EEG asistan al paciente en la recuperación de las funciones de movimiento perdidas, como fue el caso de Jan. Otro ejemplo que usa un brazo robot para asistencia es un proyecto de investigación en los Estados Unidos, donde se logró que una paciente con tetraplejia (parálisis en brazos y piernas por un daño en la médula espinal) pueda mover un brazo robot para agarrar una botella, acercarla a su boca y poder beber su contenido a través de un sorbete [1]. En el Perú, sería de suma importancia contar también con brazos robot de asistencia. Por un lado, restauraría la autonomía a personas con discapacidad, puesto que podrían realizar sus tareas diarias de forma independiente. Por otro lado, aliviaría la carga ocasionada por la falta de enfermeros particulares y privados en los hospitales del país [9].

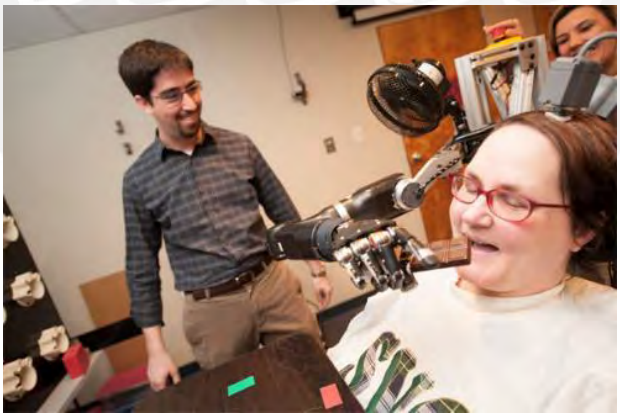


Figura 1.3. Jan Scheuermann manipulando el brazo robot para morder un chocolate. Fuente: [7]

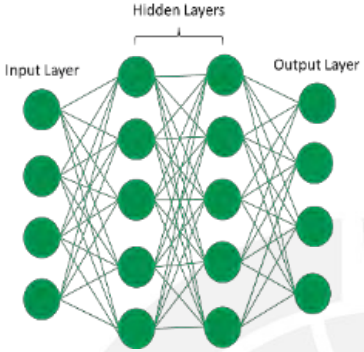
1.3. Comando de un brazo Kinova mediante BCI – Caso de estudio PUCP, Perú

La tecnología interfaz cerebro-computadora también ha sido objeto de estudio en la Pontificia Universidad Católica del Perú (PUCP). Un grupo de investigación BCI dentro de la

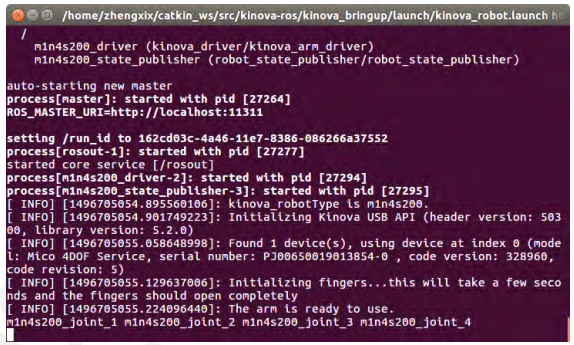
universidad ha venido desarrollando un proyecto para comandar un brazo Kinova y una silla de ruedas eléctrica a través de la activación de señales cerebrales, originada por estímulos visuales, que son detectados por un sensor EEG e interpretadas por algoritmos de aprendizaje de máquinas [10]. Con el fin de continuar las investigaciones en BCI, el presente trabajo propone también el comando de un brazo Kinova, pero el origen de la activación de las señales cerebrales se dará por estímulos de movimiento (de las extremidades superiores) y la interpretación será realizada por algoritmos de aprendizaje profundo (deep learning).

Por ello, el trabajo propuesto consta del diseño e implementación de las funciones de agarre y levante en el brazo Kinova; de esta manera, se busca plasmar las acciones de movimiento de la mano de un usuario a sus correspondientes en un brazo robot. En la Figura 1.4, se muestra un esquema del trabajo a realizar. En primer lugar, el movimiento de la mano de un usuario activará ondas cerebrales que serán detectadas por el sensor EEG. Seguidamente, un algoritmo de traducción dentro de un programa de procesamiento recibirá estas señales y enviará uno de seis eventos que corresponden a un movimiento particular de la mano. Posteriormente, el programa para comandar el brazo Kinova recibirá el evento y comandará al brazo Kinova a ejecutar la acción correspondiente a dicho evento.

Adquisición de señales EEG.
Fuente:
http://neurosky.com/2016/10/wireless-eeg-top-6-applications/



Algoritmo de traducción.
Fuente: https://devblogs.nvidia.com/wp-content/uploads/2015/03/genericDNN.png



. Algoritmo para comandar el brazo Kinova. Fuente:
https://cloud.githubusercontent.com/assets/18361752/26808504/0cd13aa2-4a12-11e7-845c-72cf103a3e88.png

Las señales electroencefalográficas obtenidas de la persona pasarán por el algoritmo de traducción. Este retornará uno de seis eventos, que serán enviados al algoritmo para comandar el Kinova, el cual enviará al brazo Kinova las posiciones de referencia correspondientes al evento recibido.

Brazo Robot Kinova Mico. Fuente:
https://www.robotsinsearch.com.au/products/mico2-robotic-arm

Figura 1.4. Propuesta de trabajo. Fuente: Fuente propia.

1.4. Objetivos de la tesis

1.4.1. Objetivo general

Diseñar e implementar la activación de las acciones de “agarre” y “levante” en el brazo robot Kinova basada en algoritmos de *deep learning* y utilizando señales EEG como entradas.

1.4.2. Objetivos específicos

A. Revisar el marco teórico: sintetizar los conceptos más importantes en tratamiento de señales y *deep learning*. Asimismo, revisar el estado del arte: realizar una búsqueda de patentes e investigaciones similares a lo propuesto.

B. Desarrollar el diseño conceptual: definir los requerimientos y limitaciones del sistema a ser desarrollado de acuerdo a las necesidades del usuario. Seguidamente, desarrollar la estructura de funciones del sistema. Finalmente, proponer tres conceptos de solución y, tras una evaluación técnico-económica, escoger un concepto de solución óptimo

C. Desarrollar el diseño preliminar: diseño de los algoritmos de traducción y comando del brazo Kinova, de la integración del sistema y de un experimento para probar resultados.

D. Desarrollar el proyecto definitivo: implementación a partir del diseño preliminar con el fin de afinar los detalles de la propuesta, y posteriormente, proceder con pruebas que confirmen el funcionamiento correcto del sistema

E. Realizar el análisis de costos del sistema integrado

1.5. Alcances de la tesis

A continuación, se tocará las limitaciones en el presente trabajo. Seguidamente, se mencionará lo que se espera como resultado al concluir el trabajo.

El entrenamiento de un algoritmo basado en *deep learning* necesita de un *dataset* con señales EEG obtenidas de un experimento con participantes ejecutando movimientos con la mano. En este trabajo, se ha decidido tomar el *dataset* obtenido de un experimento desarrollado por el consorcio WAY en Europa [4]. De este *dataset*, se utilizarán seis eventos que indican una acción de la mano: inicio del movimiento de la mano, toque del objeto con un dedo, aplicación de fuerzas, levante, reubicación del objeto y soltar objeto.

Asimismo, se empleará el brazo Kinova [11, 12], un brazo robot de 6 grados de libertad y un gripper de dos pinzas, que está disponible para uso en el CETAM, PUCP. Esta elección se debe a que el brazo Kinova cumple con las características necesarias de un robot de asistencia (adaptados para la interacción humano-robot) como, por ejemplo, bajo peso e inercia, su habilidad para funcionar con baterías de 24 VDC y sensores que previenen colisiones con los individuos.

Al término del presente trabajo, el brazo Kinova deberá moverse congruentemente correspondiendo a las entradas de las señales EEG para cada evento (acción) definido.

Capítulo 2: FUNDAMENTO TEÓRICO

En este capítulo, se realizará una revisión del marco teórico, sintetizando los conceptos más importantes en señales EEG y *deep learning*. Seguidamente, se revisará el estado del arte, en el cual se examinarán patentes e investigaciones similares a la propuesta.

2.1. Marco teórico

En esta sección, se muestra una síntesis de los conceptos y métodos que se estudian en los temas de EEG y *deep learning*; para seguidamente describir las tecnologías actuales relacionadas al tema propuesto.

2.1.1. Señales de electroencefalogramas

Las señales de encefalogramas, comúnmente llamadas ondas o ritmos cerebrales, tienen todas baja intensidad (se miden en mV), se diferencian por su frecuencia y pueden aparecer en cualquier parte del cerebro. En la Tabla 2.1, se muestran como son clasificadas comúnmente las ondas cerebrales.

2.1.2. Conceptos importantes de BCI

Desincronización Relacionada al Evento: Denominadas ERD por sus siglas en inglés (*Event-related Desynchronization*). El término fue introducido por el ingeniero, experto en biomédica, Gert Pfurtscheller para describir una reducción localizada y de corta duración en amplitud de las ondas cerebrales en la banda de frecuencia alfa [14]. Su opuesto, la sincronización relacionada al evento (ERS), describe un aumento de la amplitud en esta banda [14].

Tabla 2.1. Clasificación de las ondas cerebrales en función a su frecuencia. Fuente: [13]

Nombre de la onda	Frecuencia	Descripción
Infra-bajo	Menor a 0.5 Hz	Difíciles de medir con la tecnología actual. Se conoce poco, pero se supone que juegan un rol importante en el reloj cerebral.
Ondas Delta	Entre 0.5 y 3 Hz	Son de naturaleza lenta y se generan cuando el cuerpo duerme profundamente sin soñar. Durante este, estimulan la curación y regeneración del cuerpo humano.
Ondas Theta	Entre 3 y 8 Hz	Ocurren usualmente durante las primeras etapas del sueño y, junto a las ondas betas, durante el sueño REM
Ondas Alfa	Entre 8 y 12 Hz	Dominan cuando el cuerpo está en un estado de calma o muy poca actividad. Participan en el estado de alerta, coordinación mental y aprendizaje.
Ondas Beta	Entre 12 y 38 Hz	Se generan cuando el cuerpo interactúa activamente con el mundo externo o durante tareas cognitivas que requieren atención. Se clasifican en tres: Beta 1, Beta 2 y Beta 3. Cada fase involucra mayor actividad y energía, así como pensamientos más complejos.
Ondas Gamma	Mayores a 38 Hz	Al estar por encima de la frecuencia de disparo neuronal, se han supuesto como ruido cerebral. Sin embargo, otros investigadores lo vinculan a la percepción y a la conciencia.

Ondas Mu: Es un término utilizado en BCI para designar a las ondas que se encuentran en la misma región de frecuencia que las ondas alfa (8-12 Hz), no obstante, solo están asociadas al área sensorio motor del cerebro (córtex motor primario y córtex somato sensorial primario) [15]. Aparecen cuando el cuerpo se encuentra en estado inactivo. De esta forma, permiten detectar el movimiento cuando la actividad de ondas Mu decrece. No es necesario un movimiento físico, solo basta con imaginar el movimiento para reducir la actividad. Asimismo, cada parte del cuerpo está asociada a una región del córtex motor en la que se

activan las ondas Mu (ver Figura 2.1); en otras palabras, un movimiento de la mano, solo reducirá la actividad Mu en la zona cerebral asociada a esta parte del cuerpo.

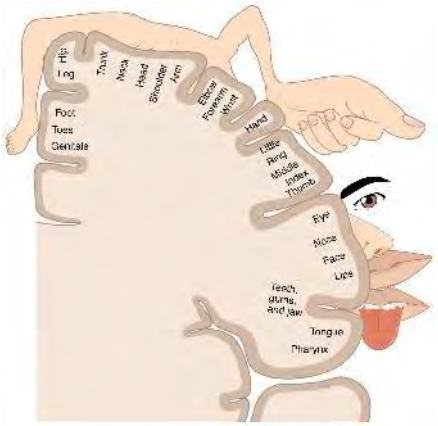


Figura 2.1. Homúnculo cortical. Cada parte del cuerpo está asociada a una región en el área sensoria motora del cerebro. Fuente: [16]

Imaginería motora: Consiste en que un individuo se imagine la realización de movimientos motores. Se utiliza para la rehabilitación luego de un paro cerebral. Asimismo, es una herramienta útil para extraer propiedades de las señales EEG en una interfaz cerebro computador.

2.1.3. Conceptos importantes de *deep learning*

En esta sección, se tocarán los conceptos más relevantes para este trabajo en los temas de estadística, *machine learning* y *deep learning*.

Common Spatial Pattern (CSP): Procedimiento matemático comúnmente utilizado para procesar señales EEG. Consiste en separar una señal en subcomponentes con el fin de maximizar la diferencia de varianza entre señales que pertenecen a distinta clase y minimizar las que pertenecen a la misma clase. Adicionalmente, permiten reducir la dimensionalidad de la información.

Event Related Potential (ERP): Se refiere a los cambios en las mediciones de los sensores EEG debido a un estímulo interno o externo, por ejemplo, la aparición de un objeto en el campo visual.

Banco de filtros: Son arreglos de filtros pasabanda, utilizados para separar una señal en componentes que pertenecen a las regiones de frecuencia de la señal original.

Machine learning: Son un conjunto de algoritmos que permiten obtener una función que permita predecir nueva información a partir de un *dataset.* Por ejemplo, predecir el precio de un auto, si se conoce su consumo de combustible, su fabricante y otros detalles técnicos.

Deep learning: Son modelos de *machine learning* basados en las redes neuronales biológicas.

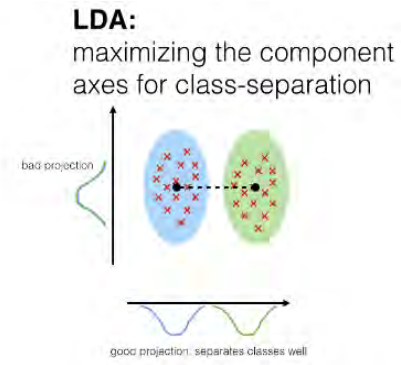
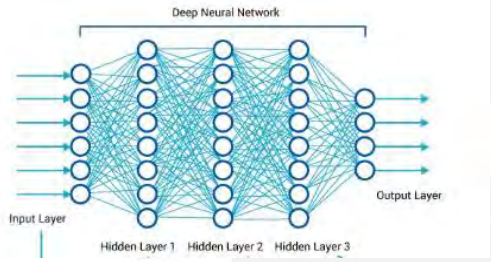
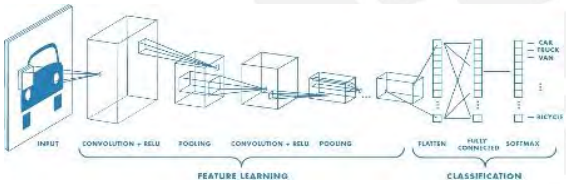
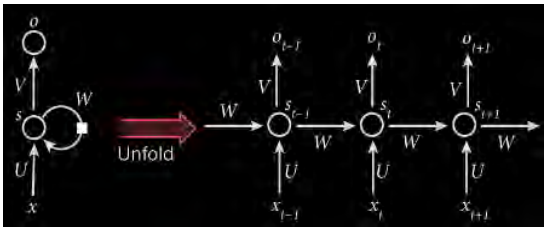
Modelos relevantes de *machine learning* y *deep learning*: Su función es ser entrenados con el fin de retornar una predicción a partir de unas entradas dadas. En la Tabla 2.3, se describen los modelos más relevantes para este trabajo.

Overfitting: Es un término utilizado para designar a los modelos que tienen un puntaje alto de predicciones correctas en los datos de entrenamiento, pero fallan cuando deben predecir con nueva información.

Bagging: Es una técnica en *machine learning* que permite reducir el *overfitting* separando los datos en múltiples cúmulos ("*bags*") y generar un modelo para cada uno. Luego, la predicción del modelo con *bagging* es el promedio de las predicciones de cada modelo generado.

Apilamiento: Es una técnica en *machine learning* que consiste en entrenar modelos distintos con los mismos datos. Posteriormente, las predicciones obtenidas son utilizadas como entradas para entrenar un modelo más (el meta-clasificador), el cual se basará en las predicciones de cada modelo para encontrar una predicción final más exacta. De esta manera, se obtiene una arquitectura como la que se observa en la Figura 2.6.

Tabla 2.3. Modelos relevantes de *machine learning* y *deep learning*. Fuente: [17, 18, 19, 20]

Nombre del modelo	Descripción
Análisis discriminante lineal  Figura 2.2. Análisis Discriminante Lineal. Fuente: [16]	Es un modelo de *machine learning* utilizado principalmente en problemas de clasificación de clases múltiples. Su objetivo es maximizar la distancia entre clases mientras minimiza la distancia dentro de la misma clase. La información de entrada debe ser gaussiana y con la misma varianza. Antes de entrenar este algoritmo, se recomienda remover valores atípicos [16].
Redes neuronales de capas múltiples  Figura 2.3. Redes neuronales de capas múltiples. Fuente: [17]	Es un modelo de *deep learning* inspirado en las redes neuronales cerebrales. Los algoritmos de redes neuronales profundas están compuestos de elementos altamente conectados (neuronas) para realizar aprendizaje no lineal [17].
Redes neuronales convolucionales  Figura 2.4. Redes Neuronales Convolucionales. Fuente: [18]	Estos modelos de *deep learning* son utilizados cuando se quiere tomar ventaja de la estructura 2D de la señal de entrada. Son muy efectivas en tareas de visión por computadora. Constan de una o más capas convolutivas seguidas por capas completamente conectadas [18].
Redes neuronales recurrentes  Figura 2.5. Redes Neuronales Recurrentes. Fuente: [19]	Estos modelos de *deep learning* reutilizan el conocimiento previamente aprendido a través de lazos internos. En otras palabras, retienen información; sin embargo, esta retención de información es sólo de corto alcance [19].

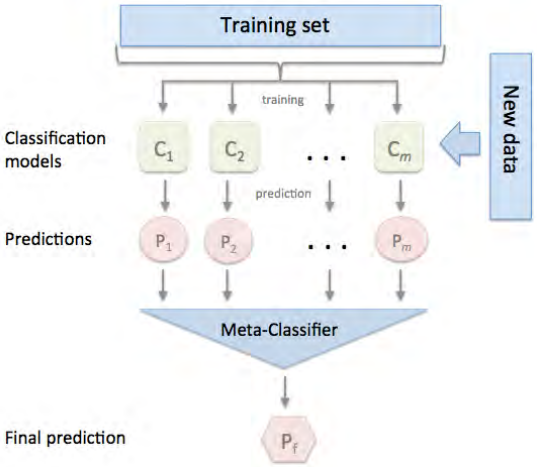


Figura 2.6. Apilamiento. Fuente: [21]

Matriz de confusión: La matriz de confusión es una tabla que describe los resultados de un clasificador o modelo. Como se observa en la Figura 2.7, existen cuatro términos por definir. El primero son los verdaderos positivos: la cantidad de veces que el modelo predijo correctamente un resultado positivo. Los verdaderos negativos son la cantidad de veces que el modelo predijo correctamente un resultado negativo. Los falsos positivos son la cantidad de veces que el modelo predijo incorrectamente un resultado positivo. Finalmente, los falsos negativos son la cantidad de veces que el modelo predijo incorrectamente un resultado negativo.

	Predicciones	
Valores reales	*Positivo*	*Negativo*
True	TP	FN
False	FP	TN

Figura 2.7. Matriz de confusión. Fuente: Fuente propia

Accuracy: Es la relación entre la cantidad de predicciones correctas hechas por el algoritmo sobre el total de predicciones.

$$Accuracy = \frac{Predicciones\ correctas}{Total\ de\ predicciones}.$$

Ratio de verdaderos positivos (TPR): Es también conocida como sensibilidad. Es la cantidad de veces que el modelo predice correctamente un resultado positivo sobre la cantidad de resultados positivos. Se calcula con la siguiente fórmula:

$$TPR = \frac{TP}{FN+TP}.$$

Ratio de falsos positivos (FPR): Es la cantidad de veces que el modelo predice de manera errónea un resultado positivo sobre el total de resultados negativos. Se calcula con la siguiente fórmula:

$$FPR = \frac{FP}{TN+FP}.$$

Área bajo la curva ROC (AUROC): Es una métrica para la clasificación binaria, que mide la utilidad del modelo en separar clases. La curva ROC se grafica calculado los ratios de verdaderos positivos y falsos positivos al cambiar el umbral de predicción del modelo. El área bajo la curva indica la separación entre clases, donde un área de 1 indica que no existe sobrelapamiento entre las clases, un área de 0.5 indica un sobrelapamiento completo, las áreas menores a 0.5 son equivalente a las mayores de 0.5 considerando que el modelo predice los positivos como negativos y estos como positivos. En la Figura 2.8, se muestra un ejemplo de una curva ROC para un modelo, la línea discontinua representa un clasificador aleatorio.

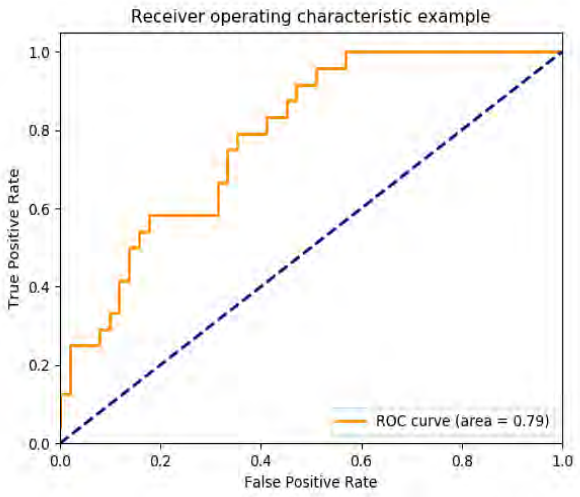


Figura 2.8. Área Bajo la Curva ROC. Fuente: [22]

2.2. Estado del arte

En esta sección, se presentarán las patentes, artículos científicos y productos comerciales más destacados respecto al trabajo presente.

2.2.1. Patentes

2.2.2.1. *BCI apparatus for stroke rehabilitation*– US 8509904B2 – Agosto 13 del 2013 [23]

Inventores: Jörn Rickert, Freiburg (DE) – Jörg Fischer, Reute (DE)

Solicitante: Cortec GmbH, Freiburg (DE)

Resumen: En la Figura 2.9, se presenta el aparato BCI cuyo objetivo es ayudar con la rehabilitación de pacientes que sufrieron de infarto cerebral. La pantalla (*display*) muestra al paciente la acción que debe realizar, luego, un electrodo EcoG envía las señales de su intención de movimiento a la unidad de evaluación de control que muestra su resultado en la pantalla.

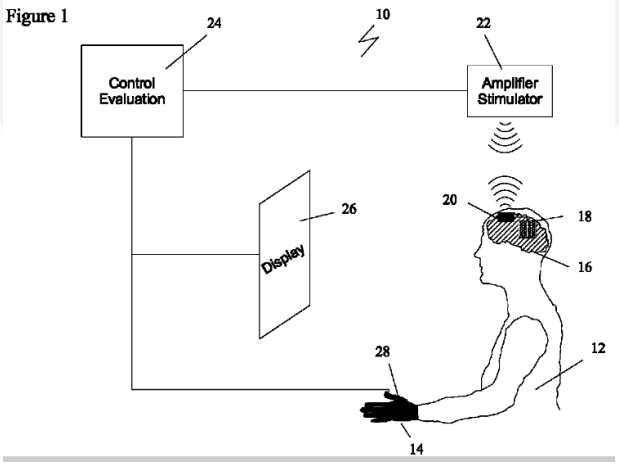


Figura 2.9. Aparato BCI para rehabilitación. Fuente: [23]

2.2.2.2. *Method and Apparatus for Biometric Analysis Using EEG and EMG Signals* – US 9155487B2 – Octubre 13 del 2015 [24]

Inventores: Michael Linderman, Ogdensburg, NY (US) – Valery I. Rupasov, Gouverneur, NY (US)

Solicitante: Norconnect Inc., Ogdensburg, NY (US)

Resumen: El método y sistema consiste en detectar, sincronizar y registrar señales EMG en los músculos de movimiento fino (ejm. la mano) y señales EEG (ver Figura 2.10); estas últimas para detectar otras biomarcas. Estas señales son guardadas y procesadas para su posterior uso en la evaluación biométrica. Esta invención es basada en el descubrimiento de los autores de que se pueden proveer biomarcas para uno o más desórdenes (ejm. Enfermedad del Parkinson).

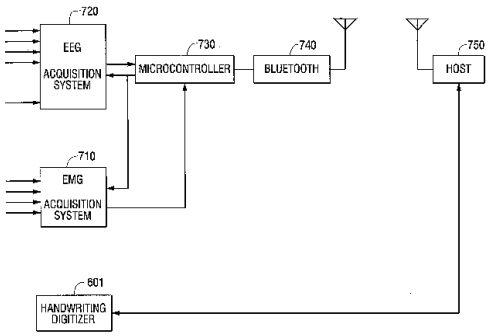


Figura 2.10. Descripción gráfica del método y aparato para análisis biométrico. Fuente: [24]

2.2.2. Artículos científicos

2.2.2.1. *Reach and Grasp by People with Tetraplegia Using a Neurally Controlled Robotic Arm* – Hochberg et al – 2012 [1]

En este estudio realizado en los Estados Unidos, se hicieron pruebas con dos participantes, a los que se les llamó S3 y T2, para demostrar su habilidad en el control neuronal de un brazo robot para realizar funciones de alcance y agarre. Para ello, ambos participantes recibieron implantes de un arreglo de microelectrodos de 96 canales ubicados en la región de la mano

dominante del córtex motor. La primera participante realizó 158 intentos en 4 sesiones y el segundo, 45 en una sola sesión. Los resultados se muestran en la Figura 2.11.

Table 1 | Summary of neurally controlled robotic arm target-acquisition trials

	Trial day 1952 S3 (DLR)	Trial day 1959 S3 (DLR)	Trial day 1974 S3 (DEKA)	Trial day 1975 S3 (DEKA)	Trial day 166 T2 (DEKA)
Number of trials	32	48	45	33	45
Targets contacted	16 (50.0%)	23 (47.9%)	34 (75.6%)	20 (60.6%)	43 (95.6%)
Grasped	7 (21.9%)	10 (20.8%)	21 (46.7%)	15 (45.5%)	28 (62.2%)
Time to touch (s)	5.4 ± 6.9	5.4 ± 2.3	6.1 ± 4.9	6.8 ± 3.6	5.5 ± 4.7
Time to grasp (s)	18.2 ± 6.4	9.5 ± 4.5	8.2 ± 4.9	8.8 ± 8.0	9.5 ± 5.5
Touched only	9 (28.1%)	13 (27.1%)	13 (28.9%)	5 (15.1%)	15 (33.3%)
Time to touch (s)	7.0 ± 6.2	4.6 ± 3.0	10.7 ± 6.5	9.4 ± 8.0	7.1 ± 6.8

Figura 2.11. Resumen de los resultados obtenidos según sesión. Fuente: [1]

En el tratamiento de señales, se utilizaron filtros Butterworth durante el proceso de adquisición de señales. Con el fin de decodificar la velocidad final del efector final del brazo y el estado de la mano de la actividad neuronal, se realizó sesiones previas en el que se les pedía a los participantes imaginar controlar el brazo robot mientras este realizaba movimientos preprogramados. Los datos obtenidos fueron guardados y, posteriormente, utilizados para entrenar un clasificador discriminante lineal. Adicionalmente, se utilizó el filtro de Kalman, con el fin de hallar con mejor fidelidad la velocidad deseada por el usuario.

Además, se realizó una prueba para estimar qué tan bien podía la participante S3 controlar el brazo DLR. La tarea consistió en agarrar una botella y acercarla al usuario para que pueda beber de ella por un sorbete (Figura 2.12). Después de un periodo de calibración y familiarización del participante con la prueba, se logró con éxito esta tarea 4 veces de 6 intentos.

Figura 2.12. Participante bebiendo de una botella con el brazo LDR. Fuente: [1]

2.2.2.2. *Noninvasive Electroencephalogram Based Control of a Robotic Arm for Reach and Grasp Task* – Meng et al – 2016 [2]

En el estudio presente, se encontró que un grupo de trece personas eran capaces de modular su actividad cerebral para comandar, con alta exactitud, un brazo robot para lograr tareas que requieren múltiples grados de libertad. Asimismo, para la detección de señales EEG se utilizó un sensor de 64 canales y las señales fueron muestreadas a 1000 Hz. Para su decodificación, se utilizó un filtro espacial laplaciano y el resultado ingresó a un modelo autoregresivo, donde se extrajo el espectro de frecuencia y mediante un mapeo lineal, la dirección de movimiento.

La tarea principal de los participantes era imaginar movimientos de sus manos derecho e izquierda, y así comandar el movimiento de un cursor y, por consiguiente, del brazo robot. Este último es el Kinova JACO y cuenta con siete grados de libertad y un *gripper* de tres dedos. En este experimento, hubo 5 fases en total: movimiento de un cursor, agarre de 1 de 4 objetos fijos, agarre de 1 de 5 objetos fijos, agarre de un objeto en una posición aleatoria, y agarre de un objeto y ponerlo en una repisa (ver Figura 2.13.). Al final de la prueba, se obtuvieron los resultados mostrados en la Tabla 2.4.

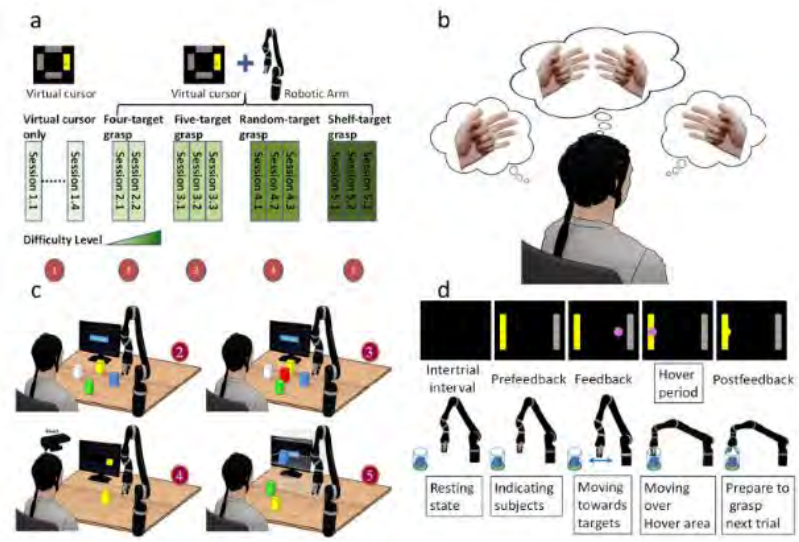


Figura 2.13. Fases 2-5 del experimento. Fuente: [2]

Tabla 2.4. Resumen de los resultados promedios obtenidos por sesión [2]

Sesión	Porcentaje válido correcto (PVC)	Tiempo promedio por cada paso (movimiento)
Agarrar 1 de 4 objetos fijos	80.3% ± 17.0%	5.5 ± 0.8 segundos
Agarrar 1 de 5 objetos fijos	77.9% ± 14.7%	5.0 ± 0.6 segundos
Agarrar 1 objeto ubicado aleatoriamente	Mayor a 62.2%, excepto con objetos ubicados en el área inferior central de la mesa de trabajo (justo enfrente del sujeto)	6.4 ± 0.7 segundos
Agarrar 1 objeto y ubicarlo en una repisa	Mayor a 71%	6.0 ± 0.5 segundos

2.2.2.3. *Evaluating Classifiers to Detect Arm Movement Intention from EEG Signals* – Planelles et al – 2014 [25]

A través de la medición de señales EEG, este paper presenta una metodología para detectar la intención de mover el brazo para alcanzar un objetivo antes de que el movimiento empiece. Este proceso se basa en tres sumas de las frecuencias espectrales de potencia involucradas en una Desincronización Relacionada al Evento. Un amplificador comercial de 16 canales y una

frecuencia de 256 Hz fueron utilizados en el experimento. Este consistía en seis participantes cuya tarea era realizar un movimiento de avance y retroceso con el mouse, y luego regresar a su posición inicial, guiados por una interfaz gráfica como se muestra en la Figura 2.14. Se probaron cuatro clasificadores: LS-SVM ("Least Squares Support Vector Machine"), QP-SVM ("Quadratic Programming Support Vector Machine"), SMO ("Sequential Minimal Optimization") y k-NN ("K Nearest Neighbors). Al concluir el estudio, se tomó como mejor opción el algoritmo LS-SVM, puesto que obtuvo el mínimo ratio de falsos positivos.

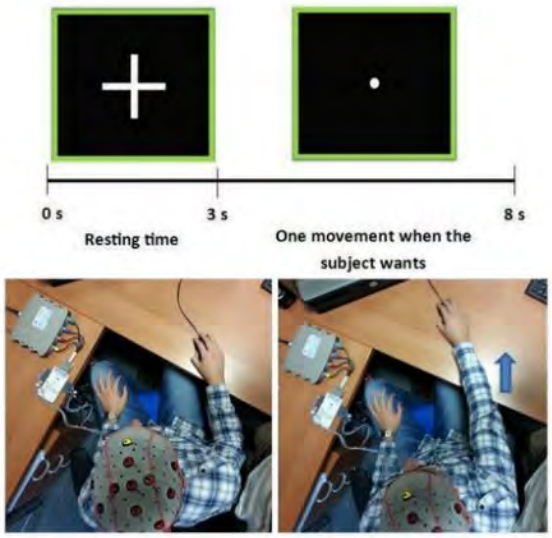


Figura 2.14. Participante realizando las tareas del experimento. Fuente: [25].

Capítulo 3: DISEÑO CONCEPTUAL

En este capítulo se describen la lista de requerimientos, la estructura de funciones y la matriz morfológica, las cuales forman parte del diseño conceptual. Usando esta información, se generarán tres conceptos de solución que serán evaluados en los aspectos técnico y económico, a fin de obtener el concepto de solución óptimo que se desarrollará como proyecto definitivo.

3.1. Lista de requerimientos

Tomando en cuenta las necesidades del usuario, se ha realizado la siguiente lista de requerimientos en la Tabla 3.1.

3.2. Estructura de funciones

En esta sección, se desarrollará la estructura de funciones del sistema. Primero, se hará una descripción general, de las entradas y las salidas del sistema. Posteriormente, se mostrará la estructura de funciones global dividida en 5 módulos: sensores, algoritmo de traducción, comando del brazo Kinova, energía y brazo Kinova. Finalmente, se describirán las funciones de cada módulo.

3.2.1. *Black Box*

El sistema (ver Figura 3.1) recibe como entrada las ondas cerebrales generadas por el usuario. A partir de ellas, se determinará el comando de acción que realizará el brazo Kinova. Es de esta manera que el sistema podrá desplazar un objeto desde una posición inicial a una final. Asimismo, el sistema necesitará energía eléctrica para su funcionamiento. Por ello, se suministra 220V AC como entrada. Las pérdidas de energía en el sistema serán principalmente por calor, luz (disipados por los dispositivos eléctricos), y ruido (generado por los actuadores mecánicos).

Tabla 3.1. Lista de requerimientos. Fuente: Fuente propia

D/E	Requerimiento
E	FUNCIÓN PRINCIPAL: Comandar el brazo robot Kinova para las funciones de agarre y levante de objetos, a través de una interfaz cerebro computador que colecta señales EEG y procesa los datos usando *deep learning*.
	GEOMETRÍA:
E	El espacio de trabajo del brazo robot Kinova cubre un volumen de 1.44 m^3.
	CINEMÁTICA:
E	Según la norma ISO 10218 para la seguridad en interacciones humano-robot, la velocidad del efector final deberá ser menor a 0.25 m/s.
	CONTROL:
D	Se desea un tiempo de respuesta del brazo* menor o igual a 5.0 ± 0.6 segundos.
D	Se desea un desempeño mayor al 75% en la métrica AUROC.
	FUERZA:
E	El brazo Kinova debe cargar un peso máximo de 0.8 kg.
	SEÑALES:
E	• Entrada: Señales de electroencefalografía producidas en el cerebro del usuario • Entrada intermedia: El evento a realizarse por el brazo Kinova representado en números del 0 al 5 • Salida intermedia: La posición angular de los motores en cada articulación del brazo Kinova y el estado (abierto o cerrado) del efector final • Salida: Efector final del brazo Kinova en posición deseada
	SOFTWARE:
D	El software que se va a utilizar debe tener idealmente las siguientes características: • código abierto • facilidad de uso • librerías de código abierto que se actualizan constantemente y adecuadamente documentadas.
D	El algoritmo debería ser fácilmente adaptable para sistemas de distinto hardware.
	COSTO:
D	El precio depende principalmente del brazo Kinova y el sensor EEG. Sumado al precio de los equipos computacionales no se deberían exceder los $50,000.

* Tiempo desde que la persona piensa el movimiento hasta que empieza a moverse el brazo robot

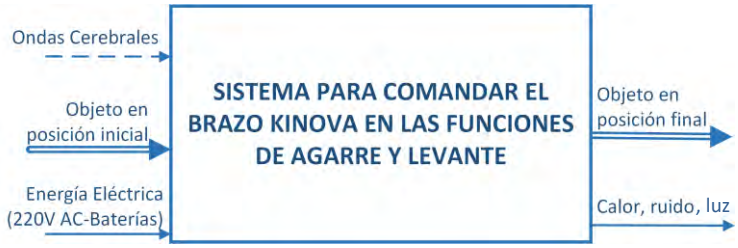


Figura 3.1. *Black Box*. Fuente: Fuente propia

3.2.2. Estructura de funciones global

En esta sección, se describen los módulos y sus funciones que componen el sistema. Seguidamente, se muestra, en la Figura 3.2, el esquema de la estructura de funciones global, agrupadas en cinco módulos, los cuales son descritos a continuación.

Sensores: Se encarga de leer las señales de entrada al sistema.

- Adquirir señal EEG: Un dispositivo registra las señales EEG producidas por las ondas cerebrales del usuario.

Algoritmo de traducción: Su objetivo es clasificar la acción que debe realizar el brazo Kinova a partir de las señales EEG y usando modelos matemáticos cuyos parámetros han sido encontrados por un algoritmo de entrenamiento.

- Procesar señal: Se procesan las señales EEG (ejm. reducción de ruido, transformaciones en frecuencia).
- Extraer características: Se obtienen las características que definen a las señales EEG. Existen tres fuentes de información: temporal, espacial y de frecuencia.
- Predecir evento: A partir de los modelos entrenados y las entradas generadas, se clasifican los eventos (acciones) que debe realizar el brazo Kinova.

Comando del brazo Kinova: Programa que permite comandar al brazo Kinova las acciones que debe realizar.

- Determinar posición angular de las articulaciones: Se determina mediante métodos matemáticos la posición angular de las articulaciones del brazo Kinova.

- Enviar posición angular de las articulaciones: Se envían señales al controlador del Kinova para mover las articulaciones a una nueva posición deseada.

Energía: Se regula el suministro de energía que alimentará a cada dispositivo.

- Energizar procesador para clasificación de eventos: Se regula el suministro de energía que alimenta el procesador que realiza la clasificación de eventos.
- Energizar procesador para el Kinova: Se regula el suministro de energía que alimenta el procesador que comanda la acción del brazo Kinova.
- Energizar brazo Kinova: Se regula el suministro de energía que alimenta al brazo Kinova.

Brazo Kinova: Representa al brazo Kinova y las acciones que efectúa.

- Accionar movimiento de las articulaciones: Los motores se mueven siguiendo los comandos del controlador Kinova.
- Accionar movimiento del efector final: Se accionan los motores que cambian de estado al efector final.
- Controlar posición angular de las articulaciones: El controlador brinda a los motores el voltaje que necesitan para moverse a la posición comandada. Asimismo, utiliza los sensores para corregir el error en posición.
- Detectar posición angular de las articulaciones: Se lee la posición de los motores y se envía la información al controlador.
- Detectar estado del efector final: Se registra el estado del efector final (porcentaje de apertura de cada dedo del *gripper*).
- Desplazar el efector final: El brazo Kinova se mueve debido a la acción de los motores.
- Sostener objeto: El efector final se cierra y sostiene el objeto.

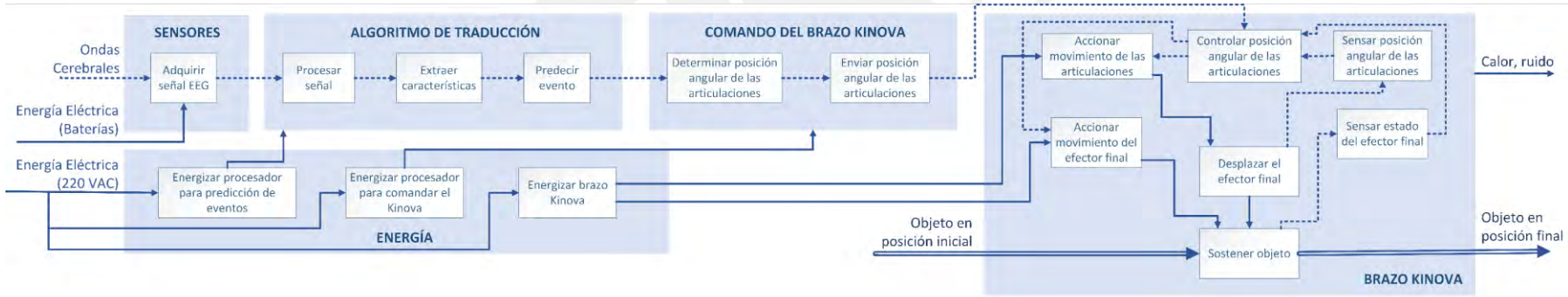
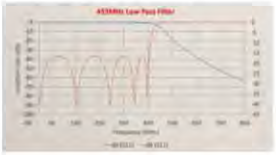
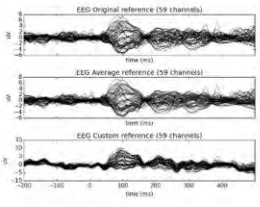


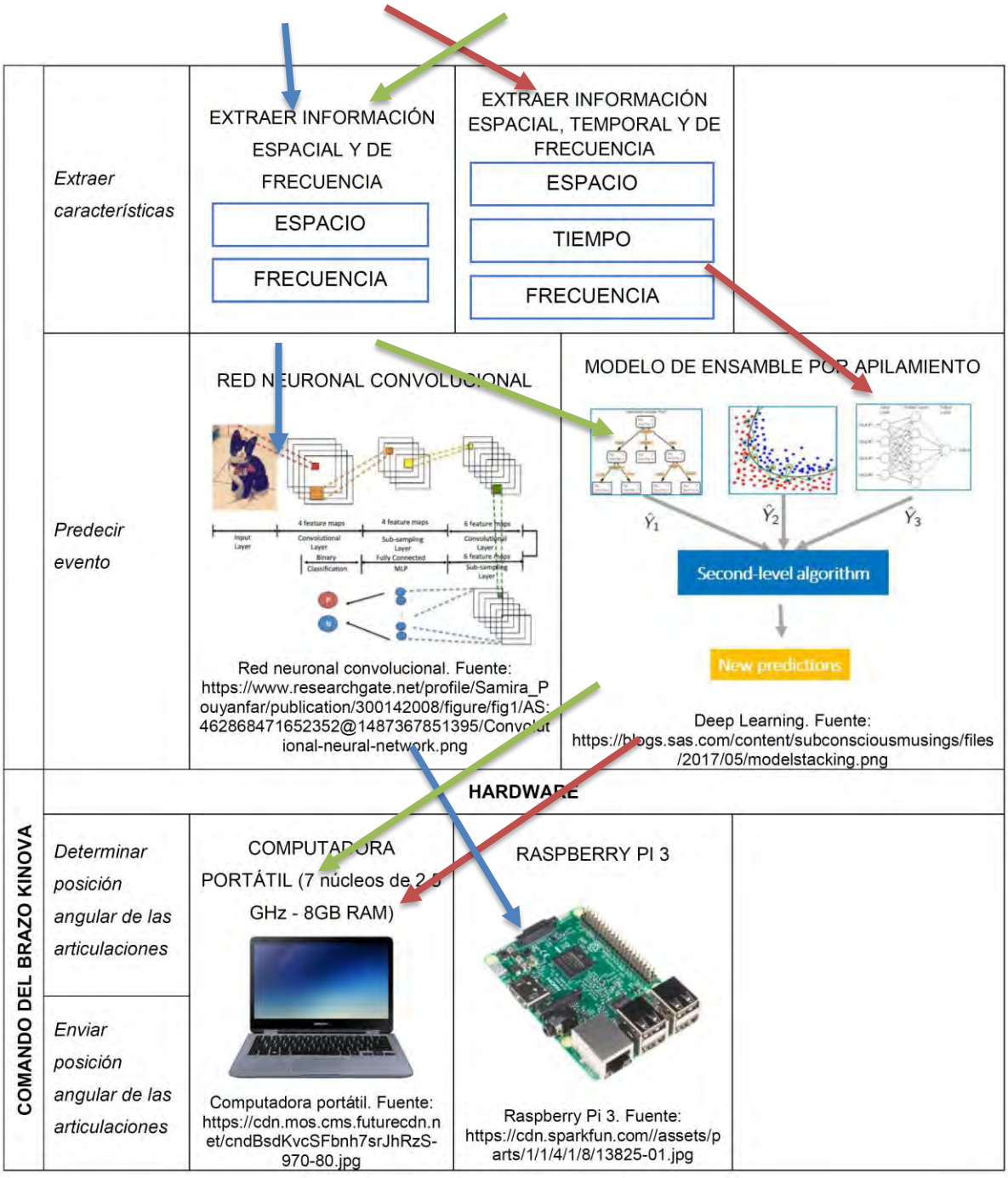
Figura 3.2. Estructura de funciones global. Fuente: Fuente propia

3.3. Matriz morfológica

A continuación, en la Tabla 3.2, se muestra la matriz morfológica del sistema.

Tabla 3.2. Matriz morfológica del sistema. Fuente: Fuente propia

	FUNCIÓN	Alternativa 1	Alternativa 2	Alternativa 3
SENSORES	*Adquirir señal EEG*	ULTRACORTEX "MARK IV" EEG HEADSET  Ultracortex "Mark IV" EEG Headset. Fuente: https://shop.openbci.com/collections/frontpage/products/ultracortex-mark-iv?variant=4356840411	G.NAUTILUS  G.Nautilus. Fuente: http://www.gtec.at/Products/Hardware-and-Accessories/g.Nautilus-PRO-Specs-Features	ACTICAP  Acticap. Fuente: http://www.bionic.es/product/acticap/
ALGORITMO DE TRADUCCIÓN		**HARDWARE**		
	Procesar señal	COMPUTADORA PORTÁTIL (7 núcleos de 2.5 GHz - 8GB RAM)  Computadora portátil. Fuente: https://cdn.mos.cms.futurecdn.net/cndBsdKvcSFbnh7srhRzS-970-80.jpg	RASPBERRY PI 3 + SERVIDOR  Raspberry Pi 3. Fuente: https://cdn.sparkfun.com//assets/parts/1/1/4/1/8/13825-01.jpg	 Servidor. Fuente: https://i0.wp.com/beebom.com/wp-content/uploads/2017/02/Raspberry_pi_alternatives_1-1.jpg?w=496&ssl=1
	Extraer características			
	Predecir evento			
ALGORITMO DE TRADUCCIÓN		**SOFTWARE**		
	Procesar señal	BANCO DE FILTROS PASABAJO  Banco de filtros. Fuente: http://awgrf.com/low-pass-filter-uhf-transmitter/	BANCO DE FILTROS + RE-REFERENCIAR  Banco de filtros y re-referenciar. Fuente: https://martinos.org/mne/stable/_images/sphx_glr_plot_rereference_eeg_001.png	

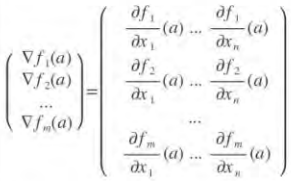
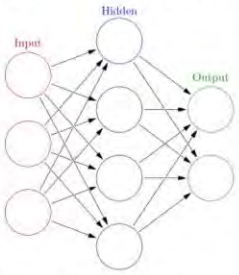


Red neuronal convolucional. Fuente: https://www.researchgate.net/profile/Samira_Pouyanfar/publication/300142008/figure/fig1/AS:462868471652352@1487367851395/Convolutional-neural-network.png

Deep Learning. Fuente: https://blogs.sas.com/content/subconsciousmusings/files/2017/05/modelstacking.png

Computadora portátil. Fuente: https://cdn.mos.cms.futurecdn.net/cndBsdKvcSFbnh7srJhRzS-970-80.jpg

Raspberry Pi 3. Fuente: https://cdn.sparkfun.com//assets/parts/1/1/4/1/8/13825-01.jpg

<table>
<tr><td rowspan="3">COMANDO DEL BRAZO KINOVA</td><td colspan="4">SOFTWARE</td></tr>
<tr><td>Determinar posición angular de las articulaciones</td><td>MÉTODO DE LA TRANSPUESTA JACOBIANA
 
Transpuesta Jacobiana. Fuente: http://www.ub.edu/glossarimatec o/sites/default/files/imatges_fitxe s/imagenes%20Funciones/Matri z%20jacobiana.PNG</td><td>FABRIK
 
FABRIK. Fuente: http://www.andreasaristidou.com/p ublications/images/FABRIK.png</td><td>REDES NEURONALES
 
Redes Neuronales. Fuente: https://upload.wikimedia.org/wikip edia/commons/thumb/4/46/Colore d_neural_network.svg/296px-Colored_neural_network.svg.png</td></tr>
<tr><td>Enviar posición angular de las articulaciones</td><td>ROS
 
ROS. Fuente: https://ricveal.com/blog/ros-framework/</td><td>KINOVA SDK (WINDOWS)
 
Kinova SDK (Windows). Fuente: http://ramalco.co.uk/portfolio/softw are/</td><td></td></tr>
<tr><td>ENERGÍA</td><td>Energizar procesador para predicción de eventos</td><td>BATERÍA PARA LAPTOP
 
Batería para Laptop. Fuente: https://www.laptopbatteryexpres s.com/v/vspfiles/photos/DELL-73-Vostro-1015-2.jpg</td><td>TRANSFORMADOR PARA ALIMENTAR SBC
 
Transformador para SBC. Fuente: https://www.raspberrypi.org/magpi /wp-content/uploads/2017/04/Power-supply.jpg</td><td></td></tr>
</table>

ENERGÍA	*Energizar procesador para comandar el Kinova*	BATERÍA PARA LAPTOP  Batería para Laptop. Fuente: https://www.laptopbatteryexpress.com/v/vspfiles/photos/DELL-7[illegible]-Vostro-1015-2.jpg	TRANSFORMADOR PARA ALIMENTAR SBC  Transformador para SBC. Fuente: https://www.raspberrypi.org/magpi/wp-content/uploads/2017/04/Power-supply.jpg	
	Energizar brazo Kinova			
BRAZO KINOVA	*Controlar posición angular de las articulaciones*	BRAZO KINOVA  Brazo Robot Kinova Mico (6 GDL). Fuente: Kinova Inc., "MICO2 User guide". p. 39, 2017.		
	Accionar movimiento de las articulaciones			
	Accionar movimiento del efector final			
	Sensar posición angular de las articulaciones			
	Sensar estado del efector final			
	Desplazar el efector final			
	Sostener objeto			

SOLUCIÓN 1

SOLUCIÓN 2

SOLUCIÓN 3

3.4. Conceptos de solución

Concepto de solución 1:

En la Figura 3.3, se presenta el primer concepto de solución, el cual utiliza el G.Nautilius. Este tiene mayor precisión que el Ultra Cortex Mark IV, sin ser tan costoso como el Acticap. Para el procesamiento del algoritmo de traducción, se utiliza una laptop, cuya fuente de alimentación es su propia batería. Esto es una opción menos costosa comparada con el uso de un servido que realice la computación. Las señales EEG se procesan con un banco de filtros, lo que permite obtener varias frecuencias necesarias para la extracción de características. Para estas, también se utilizará información en el dominio espacial y de frecuencia en una cantidad definida de ventanas de tiempo. Estas características se utilizarán en un modelo de conjunto que utiliza la técnica de apilamiento para mejorar la clasficación de varios modelos de *deep learning*.

El procesamiento del algoritmo para comandar al brazo Kinova se realizará en una computadora portátil, cuya fuente de alimentación será una batería. El cálculo de las posiciones angulares se realizará por el método de la transpuesta Jacobiana (método tradicional para cinemática inversa), y la comunicación con el controlador del brazo Kinova se realizará por medio de ROS, que ya ofrece un paquete de librerías para esta aplicación.

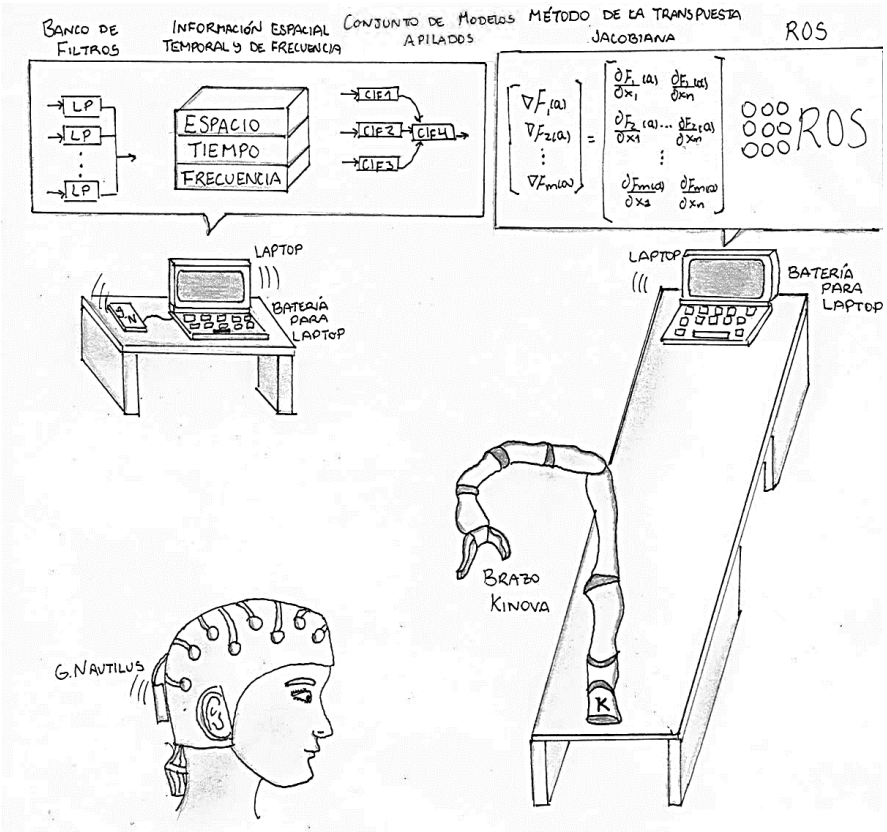


Figura 3.3. Concepto de solución 1. Fuente: Fuente propia.

Concepto de solución 2:

La Figura 3.4 muestra el segundo concepto de solución, el cual utiliza el Ultra Cortex Mark IV, la opción menos costosa entre los sensores EEG, no obstante, la de menor precisión. Para el procesamiento del algoritmo de traducción, se utiliza una computadora portátil, cuya fuente de alimentación será una batería. El uso de un servidor hubiera obtenido resultados con mayor rapidez, pero el costo sería mayor. El procesamiento de las señales EEG, se realizará mediante un banco de filtros. Así obtener la señal en frecuencias necesarias para la extracción de características; la cual, se realizará en el dominio espacial y de frecuencia. En la clasificación de eventos utilizando estas características, se utilizará una red neuronal convolucional como clasificador.

Para el procesamiento del algoritmo para comandar al brazo Kinova, se utilizará un Raspberry Pi 3, cuya fuente de alimentación será un circuito de alimentación con batería. El cálculo de las velocidades angulares se realizará a través de redes neuronales, que permiten un cómputo sencillo; y la comunicación con el controlador del brazo Kinova será por medio de ROS, que ya ofrece un paquete de librerías para esta aplicación.

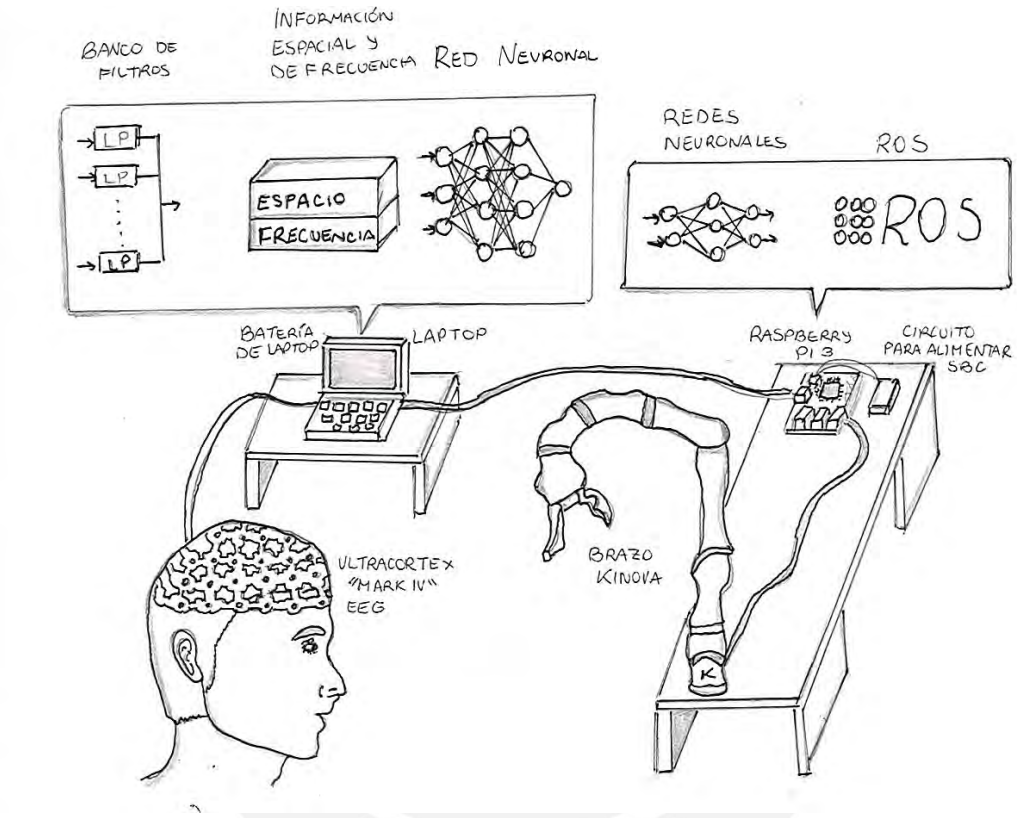


Figura 3.4. Concepto de solución 2. Fuente: Fuente propia.

Concepto de solución 3:

El tercer concepto de solución se muestra en la Figura 3.5. Se utiliza el Acticap, que cuenta con la mayor cantidad de electrodos secos y mejor precisión. Para el procesamiento del algoritmo de traducción, se utiliza un Raspberry Pi 3 para obtener los datos del sensor EEG. Luego, el microcontrolador enviará estos datos un servidor, donde se realizará la computación. La fuente de alimentación del Raspberry Pi 3 será energía acondicionada por

un transformador. Las señales EEG se procesan con un banco de filtros y el método denominado re-referenciamiento. Esto permite un buen filtrado de la señal y el re-referenciamiento ayuda a tener una mejor muestra de entrada para la extracción de características. Estas utilizarán información en el dominio espacial y de frecuencia. La clasificación de eventos, a partir de estas características, será realizada por un conjunto de modelos utilizando la técnica de apilamiento con el fin de mejorar la clasificación de cada modelo.

De igual manera, se utilizará una laptop, cuya fuente de alimentación será una batería, para el procesamiento del algoritmo de clasificación de eventos. El cálculo de las velocidades angulares se realizará a través de FABRIK, un método heurístico para calcular cinemática inversa. La comunicación con el controlador del brazo Kinova se realizará por el Kinova SDK en Windows, un programa desarrollado por el mismo Kinova para facilitar el uso de los brazos robot que fabrica.

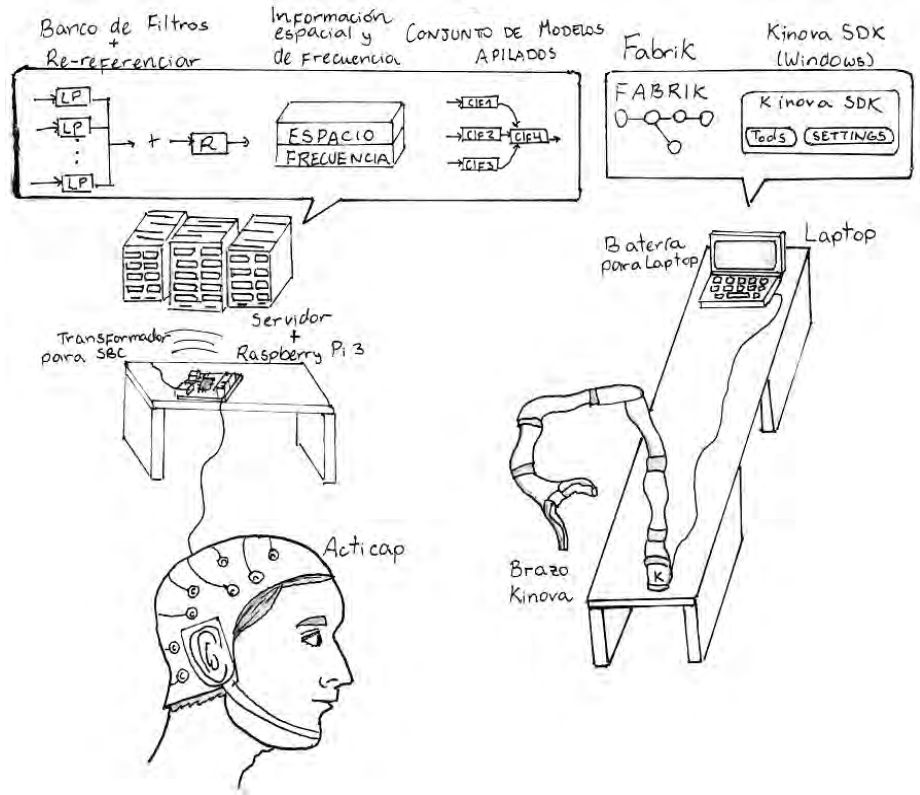


Figura 3.5. Concepto de solución 3. Fuente: Fuente propia.

3.4.1. Evaluación técnico-económica

A continuación, se presenta en la Tabla 3.3, Tabla 3.4 y en la Figura 3.6, la evaluación técnico-económica de los conceptos de solución antes descritos.

Tabla 3.3. Evaluación técnica. Fuente: Fuente propia

Criterios Técnicos	Pesos g	Proyectos							
		Solución 1		Solución 2		Solución 3		Solución Ideal	
		p	g*p	p	g*p	p	g*p	p	g*p
Facilidad para utilizar	3	3	9	2	6	1	3	4	12
Seguridad	3	2	6	1	3	2	6	4	12
Confiabilidad	4	3	12	1	4	3	12	4	16
Complejidad	1	3	3	2	2	2	2	4	4
Transportabilidad	1	2	2	3	3	1	1	4	4
Costo computacional de los algoritmos	2	2	4	3	6	3	6	4	8
Velocidad de procesamiento de los equipos	2	3	6	2	4	2	4	4	8
Total		18	42	14	28	14	34	28	64
Valor técnico		0.66		0.44		0.53		1	

Tabla 3.4. Evaluación económica. Fuente: Fuente propia

Criterios Económicos	Pesos g	Proyectos							
		Solución 1		Solución 2		Solución 3		Solución Ideal	
		p	g*p	p	g*p	p	g*p	p	g*p
Grado de contaminación al medio ambiente	2	2	4	2	4	2	4	4	8
Fácil adquisición de los componentes	3	3	9	2	6	1	3	4	12
Costo de la tecnología	4	2	8	3	12	1	4	4	16
Total		7	21	7	22	4	11	12	36
Valor económico		0.58		0.61		0.31		1	

Selección del Concepto de solución óptimo:

En la Figura 3.6, se observa que el concepto de solución 1 muestra mejor desempeño que el tercero. Respecto al concepto de solución 2, este muestra el mejor resultado en una evaluación económica. A pesar de ello, este concepto se desempeña insatisfactoriamente en la parte técnica, por lo que se toma como solución óptima al concepto de solución 1.

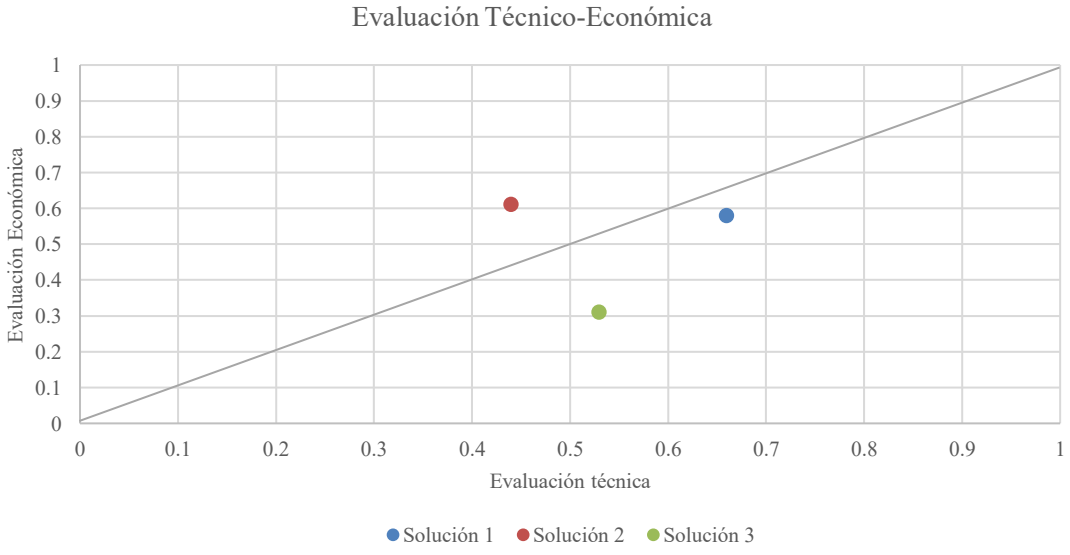


Figura 3.6. Evaluación técnico económica. Fuente: Fuente propia

Capítulo 4: DISEÑO PRELIMINAR

El presente capítulo tratará la etapa de diseño preliminar, la cual se aborda en cinco secciones. Se inicia con una descripción del diagrama de bloques del sistema de control. Seguidamente, se trata el diseño del algoritmo que interpreta las señales EEG (algoritmo de traducción) y el diseño del algoritmo que comanda acciones al brazo Kinova. Posteriormente, se presenta la integración de los módulos, y finalmente, el diseño del experimento a realizarse con usuarios.

4.1. Diagrama de bloques del sistema de control

El trabajo propone dos componentes principales del sistema de control: el primero es el algoritmo de traducción, el segundo, el comando del brazo Kinova. En la Figura 4.1, se muestran los programas correspondientes con sus entradas y sus salidas. Asimismo, se detalla información de las entradas, salidas y los eventos. Si bien los datos de los cuales se clasificarán resultados pertenecen a un *dataset* (usado también para el entrenamiento), el trabajo presente también propone el planteamiento de un experimento, con el fin de poder probar el desempeño del algoritmo de traducción bajo condiciones más cercanas a la realidad. En las siguientes secciones, se mostrarán ambos programas con mayor detalle.

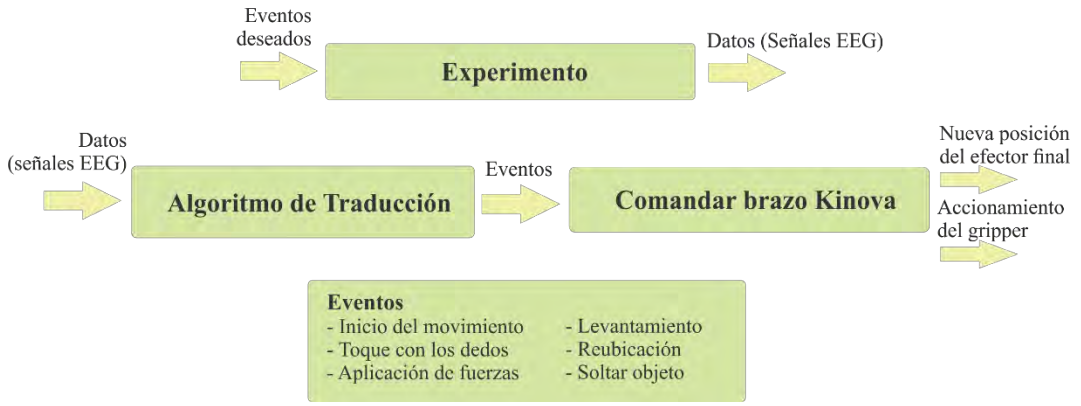


Figura 4.1. Diagrama de bloques del sistema de control (lazo abierto) Fuente: Fuente propia

4.2. Algoritmo de traducción: Diseño

El algoritmo de traducción se encarga de interpretar las señales EEG y encontrar la acción que debe efectuar el programa que comanda el brazo Kinova. Un conjunto de modelos apilados se encarga de la interpretación mencionada. El proceso de entrenamiento consiste en hallar todos los parámetros (pesos) del modelo matemático a partir de entradas y salidas obtenidas a través de un experimento previamente realizado. La etapa de diseño implica formular un algoritmo que utiliza el modelo matemático con el fin de hallar una salida a partir de una entrada.

4.2.1. Datos para entrenamiento de modelos

El entrenamiento de los modelos requiere de entradas cuyas salidas se conocen. Para obtener estos datos, se realizan pruebas experimentales. Para el presente trabajo, se ha seleccionado un experimento financiado por la Unión Europea en su programa WAY (*Wearable Interfaces for hAnd recoverY*) [5].

El nombre de la base de datos es WAY-EEG-GAL y cuenta con registros de EEG para la investigación de técnicas de decodificación de sensaciones, intenciones y acciones en las funciones de agarre y levante. La adquisición de datos fue realizada por una gorra EEG y un amplificador de señales, cuyo ratio de muestreo en software fue de 500 Hz. Antes de cada experimento, a cada participante recibió las instrucciones mostradas en la Figura 4.2.

Sit close to the table, relax your shoulder and place your upper arm next to your body. The elbow joint shall be higher than the wrist. During performance of the task, the forearm shall not touch the table. Your left arm should rest close to your waist. The red light is the signal to reach out and lift the object. Grasp the object with your thumb and index finger, in the middle of the grey surface and lift the object about 5 cm from the table. You should lift the object into the circle and hold it there until the red light turns off. Place the object on the table and place your arm next to your body. You shall rest your hand on the 'blue surface'—relax your shoulder.

Figura 4.2. Instrucciones del experimento para la generación de la base de datos WAY-EEG-GAL. Fuente: [5]

En la Figura 4.3, se observa al participante en una de las pruebas. Obsérvese que los datos que servirá como entrada se obtiene solo de las señales EEG, mostradas en la Figura 4.4; mientras que, por otro lado, los sensores EMG coleccionan datos que permitirán inferir la salida. Existen más de 16 eventos tomados, pero para el trabajo presente nos interesan solamente seis: el inicio del movimiento desde el reposo, el toque del objeto, la aplicación de fuerza para levantarlo, el levantamiento, la reubicación y la acción de soltar el objeto.

Figura 4.3. Participante realizando el experimento para generación de la base de datos WAY-EEG-GAL. Fuente: [5]

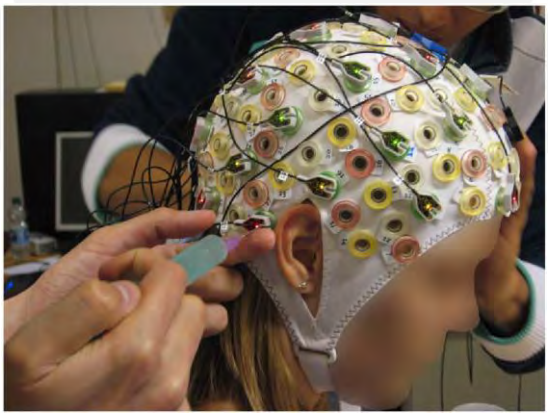


Figura 4.4. Sensor EEG utilizado para el experimento. Fuente: [5]

El experimento involucraba cinco tipos de series usando el artefacto presentado en la Figura 4.5 para modificar las condiciones del experimento. El primer tipo se efectuaba a manera de práctica, con el fin de familiarizar al participante con la tarea. El segundo, consistía en levantar 34 veces el objeto con cambios impredecibles de peso (165, 330 y 660 g). El tercero,

consistía, nuevamente, en levantar 34 veces el objeto, pero con superficies de fricción variadas (lija, gamuza y seda). El cuarto, 28 levantamientos con cambios insospechados en el peso y la fricción. El quinto consistía en que los participantes debían separar los dedos hasta que un deslizamiento del objeto ocurría. De las cinco, solo la segunda, tercera y cuarta tienen registros EEG; por ello, son las que se utilizarán para el presente trabajo. En total, para cada participante, se grabaron 10 series (6 series de la segunda, 2 de la tercera y 2 de la cuarta), de las cuales, este trabajo utiliza 8 de ellas (4 de la segunda, 2 de la tercera y 2 de la cuarta), puesto que 2 no tienen información de los eventos. En promedio, cada serie tiene 188008 muestras, equivalentes a una duración de 376s contenidos en archivos de 27MB. De las 8 series, 6 se utilizarán con el fin de entrenar los modelos para el algoritmo de traducción; las siguientes 2, serán usadas para la validación del algoritmo de traducción con los modelos ya entrenado.

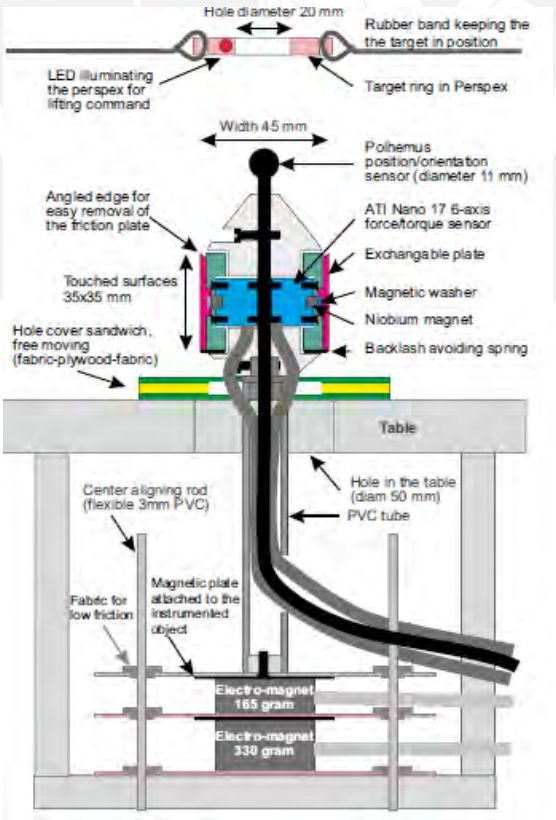


Figura 4.5. Objeto y demás artefactos utilizados en el experimento para la generación de la base de datos WAY-EEG-GAL. Fuente: [5]

4.2.2. Algoritmo para entrenamiento de modelos

El algoritmo de entrenamiento fue basado en un desarrollo de Alexander Barachant y Rafal Cycon [3]. En la Figura 4.6, se muestra el esquema de niveles utilizado en el programa. A esta técnica de agrupar modelos por niveles se denomina apilamiento. En un conjunto de modelos apilados, las clasificaciones de un nivel son las entradas del siguiente nivel. En el nivel de preprocesamiento, se emplea un banco de filtros, para extraer la información de frecuencia; matrices de covarianza, para información espacial; y potenciales relacionados al evento, para la información temporal. En el nivel 1, se encuentran modelos que utilizan regresión logística y análisis de discriminante lineal, pero también incluyen modelos de redes neuronales recurrentes (RNN) y convolucionales (CNN). En el programa, los modelos de nivel 1 se agrupan con las funciones de procesamiento y resultan en 50 modelos, que se diferencian en sus parámetros de procesamiento y/o arquitectura. En el segundo nivel, se encuentran cuatro modelos del tipo *extreme gradient boosting* (XGB). Algunos de estos modelos se entrenan utilizando la técnica *bagging* y otros sin ella. Para el presente trabajo, se trabajará con 4 de estos modelos. En el último nivel, se emplean las medias ponderadas aritméticas, geométrica y de potencia: $\frac{1}{1+e^{-\sum X^w}}$ Por consiguiente, las clasificaciones de algunos modelos tendrán más peso que otros en la clasificación final, la cual será el promedio de las tres medias mencionadas. En la Figura 4.7, se muestra en diagrama de flujo la secuencia del programa. Si bien el programa referido contiene las arquitecturas de los 55 modelos en archivos yaml, los pesos o "*weights*" que complementan estas arquitecturas no son guardados. Es por ello que se han efectuado modificaciones al programa original, todas ellas relacionadas al guardado de los pesos de los modelos para el procesamiento y la clasificación. Los valores de AUROC de este entrenamiento para los modelos de nivel 2 y 3 se detallan en la Tabla 4.1. Adicionalmente, se ha procedido a ejecutar el entrenamiento en un

procesador Xeon de doce núcleos de 3.5GHz. El detalle de cada modelo se explica en la siguiente sección.

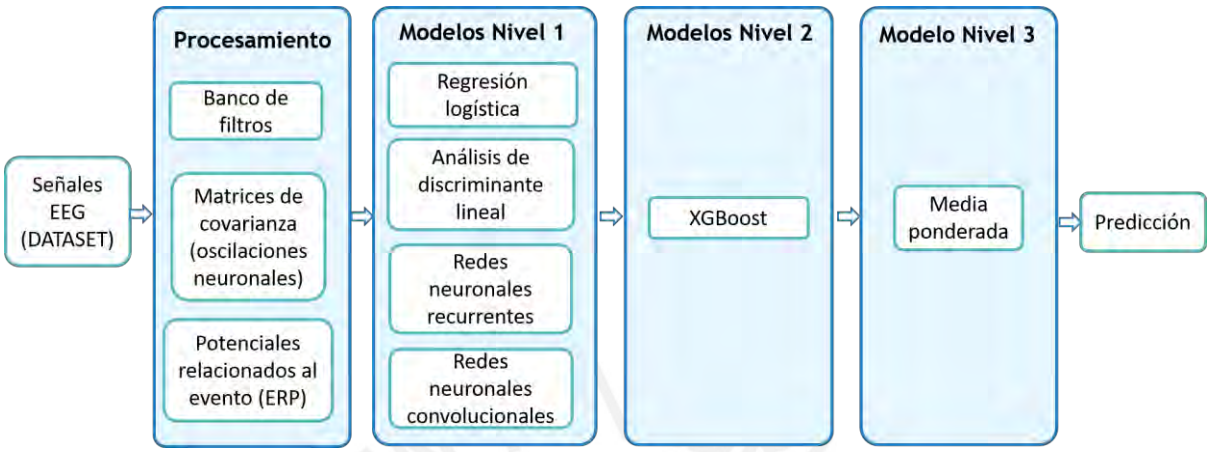


Figura 4.6. Esquema de tres niveles utilizado en el concurso Grasp & Lift EEG. Fuente: Fuente propia, basado en [6]

Tabla 4.1. Resultados de AUROC para los modelos en la capa 2 y 3. Fuente: Fuente propia

Modelos	AUROC
XGB (bags 15, size 8, depth 5)	0.965
XGB (bags 15, size 25, depth 5)	0.962
XGB (bags 15, size 8, depth 5, delay 1000)	0.957
XGB (delay 150)	0.954
Media ponderada (nivel 3)	0.970

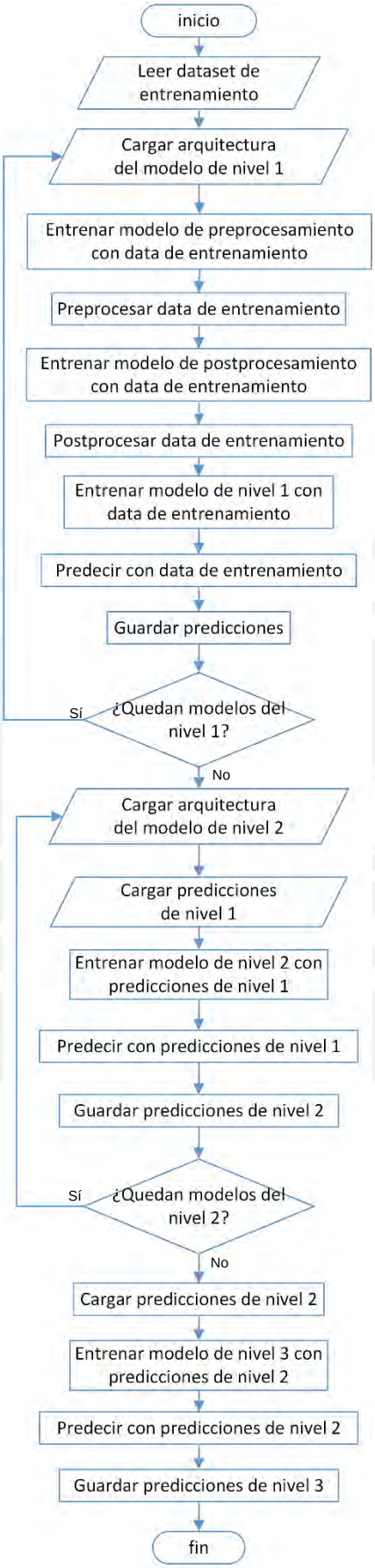


Figura 4.7. Diagrama de flujo del programa de entrenamiento. Fuente: Fuente propia

4.2.3. Detalles de los modelos entrenados

Los modelos del algoritmo de traducción de nivel 1 se pueden clasificar en cinco tipos: los de covarianza, banco de filtros, CNN y RNN.

Modelos de Covarianza: Estos modelos calculan las matrices de covarianza basándose en la técnica *Common Spatial Patterns* con la excepción de que, en lugar de hacer cálculos de distancia en geometría euclidiana, estos se basan en geometría de Riemann; en consecuencia, se consideran a las propias matrices como pertenecientes a un espacio propio [26]. Asimismo, estos modelos se pueden clasificar en dos tipos, dependiendo de la manera que se agrupan los eventos. En un caso, se utilizan 7 eventos, que incluyen los 6 eventos por detectar más uno que representa ningún evento encontrado. En el otro caso, se utilizan 12 eventos, en los que cada evento puede ser detectado como existente (clase 1) o no existente (clase 0), lo que permite concurrencia de los eventos. Adicionalmente a estos dos tipos de modelo, existe un modelo que calcula las matrices de covarianza, no a partir de la *técnica Common Spatial Patterns*, sino estimando el promedio de los Potenciales Relacionados al Evento (ERP) de todos los sucesos de una clase dada. Posteriormente, las matrices resultantes se concatenan con las matrices de covarianza antes de la estimación [27].

Previamente a este paso, los datos han sido preprocesada por un banco de filtros, submuestreada y se han tomado ventanas con una cierta cantidad de puntos en el tiempo como entradas para el cálculo de la covarianza. Posteriormente, los *features* resultantes del procesamiento y la reducción de dimensionalidad pasan por un clasificador *Linear Discriminant Analysis* (LDA) o de Regresión Logística. Adicionalmente, existe un modelo que toma el procesamiento de todos los modelos de este tipo y los utiliza para entrenar un clasificador de Regresión Logística. Los parámetros y las funciones de procesamiento para cada modelo se detallen en la Tabla 4.2.

Tabla 4.2. Parámetros de los modelos tipo covarianza. Fuente: Fuente propia

Modelo	Filtros de corte	Ventana	Submuestreo	Num Eventos	Clasificador
Cov7 500 1-15 LR	1 Hz, 15 Hz	500	10	7	Regresión Logística
Cov7 500 1-15 LDA	1 Hz, 15 Hz	500	10	7	LDA
Cov7 500 1-15 PLR	1 Hz, 15 Hz	500	10	7	Regresión Logística Polinomial (Grado 2)
Cov7 500 7-30 LR	7 Hz, 30 Hz	500	10	7	Regresión Logística
Cov7 500 7-30 LDA	7 Hz, 30 Hz	500	10	7	LDA
Cov7 500 7-30 PLR	7 Hz, 30 Hz	500	10	7	Regresión Logística Polinomial (Grado 2)
Cov7 500 20-35 LR	20 Hz, 35 Hz	500	10	7	Regresión Logística
Cov7 500 20-35 LDA	20 Hz, 35 Hz	500	10	7	LDA
Cov7 500 20-35 PLR	20 Hz, 35 Hz	500	10	7	Regresión Logística Polinomial (Grado 2)
Cov7 500 70-150 LR	70 Hz, 150 Hz	500	10	7	Regresión Logística
Cov7 500 70-150 LDA	70 Hz, 150 Hz	500	10	7	LDA
Cov7 500 70-150 PLR	70 Hz, 150 Hz	500	10	7	Regresión Logística Polinomial (Grado 2)
Cov7 250 35 LR	35 Hz	250	10	7	Regresión Logística
Cov7 250 35 LDA	35 Hz	250	10	7	LDA
Cov7 250 35 PLR	35 Hz	250	10	7	Regresión Logística Polinomial (Grado 2)
Cov7 500 35 LR	35 Hz	500	10	7	Regresión Logística
Cov7 500 35 LDA	35 Hz	500	10	7	LDA
Cov7 500 35 PLR	35 Hz	500	10	7	Regresión Logística Polinomial (Grado 2)
Cov12 256 35 LR	35 Hz	256	10	12	Regresión Logística
Cov12 500 35 LR	35 Hz	256	10	12	Regresión Logística
Cov ERP LR	1 Hz, 20 Hz	500	10	-	Regresión Logística
Cov ERP PLR	1 Hz, 20 Hz	500	10	-	Regresión Logística Polinomial (Grado 2)
Cov All	-	-	-	-	Regresión Logística

Modelos con Banco de Filtros: El único procesamiento de los datos para estos modelos son filtros pasabanda o pasabajo; sin embargo, hay dos subtipos en este tipo de modelo que además de los datos preprocesados por el banco de filtros, usan como *features* los modelos de covarianza "Cov7 250 35" y "Cov12 256 35". Asimismo, igual que con los modelos de covarianza, hay un subtipo que toma el resultado de todos los procesamientos de los modelos de covarianza y los de banco de filtros, y los utiliza para entrenar un clasificador, el cual puede ser el de Regresión Logística o Linear Discriminant Analysis. Adicionalmente, se utilizan tres técnicas de normalización: normalización L1, normalización L2 y estandarización; las cuales varían para cada modelo. A continuación, en la Tabla 4.3 se muestran los parámetros para cada modelo de este tipo.

Tabla 4.3. Parámetros de los modelos tipo banco de filtros. Fuente: Fuente propia

Modelo	Filtros de corte	Normalización	Clasificador
FBL_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	Regresión Logística
FBL_L2	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L2	Regresión Logística
FBL_SC	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	Estandarizar	Regresión Logística
FBL_LDA	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	-	LDA
FBL_LDA_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	LDA
FBL_Cov7_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	Regresión Logística
FBL_Cov7_L2	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L2	Regresión Logística
FBL_Cov7_SC	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	Estandarizar	Regresión Logística
FBL_Cov7_LDA	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	-	LDA
FBL_Cov7_LDA_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	LDA

FBL_Cov12_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	Regresión Logística
FBL_Cov12_L2	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L2	Regresión Logística
FBL_Cov12_SC	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	Estandarizar	Regresión Logística
FBL_Cov12_LDA	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	-	LDA
FBL_Cov12_LDA_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	LDA
FBL_delay_L1	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L1	Regresión Logística
FBL_ delay _L2	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	L2	Regresión Logística
FBL_ delay _SC	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	Estandarizar	Regresión Logística
FBL_ delay _LDA	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	-	LDA
FBL_All_LR	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	SC	Regresión Logística
FBL_All_LDA	0.5 Hz, 1 Hz, 2 Hz, 3 Hz, 4 Hz, 5 Hz, 7 Hz, 9 Hz, 15 Hz, 30 Hz	SC	LDA

Modelos Convolucionales: En los modelos convolucionales, tenemos dos tipos, los que utilizan una capa convolucional 2D y los que utilizan una capa convolucional 1D. Los primeros construyen la información como una matriz de 1 solo canal con 32 filas correspondiente a los 32 electrodos y 500 columnas, valor que corresponde a la cantidad de puntos en 1 segundo para una frecuencia de 500 Hz. En cambio, en la capa convolucional 1D, la matriz tiene 32 canales correspondiente a los 32 electrodos y una longitud temporal de 500 puntos en el tiempo. La arquitectura para ambos tipos de modelos tiene gran similitud como se muestra en las Figura 4.8 y Figura 4.9. En ambos, a la entrada le continúa la capa convolucional, seguida por tres capas completamente conectadas (*fully connected)* de 1024 nodos, en todas las cuales, la función de activación es de tipo Relu. El término *dropout*,

mostrado también en la imagen, no es una capa, sino una técnica que permite reducir el *overfitting*, un problema común en modelos CNN. Adicionalmente, para ambos tipos de modelos, el preprocesamiento se constituye de un banco de filtros y submuestreo. Los parámetros del preprocesamiento varían para cada modelo CNN y se pueden encontrar en la Tabla 4.4. Para el entrenamiento, se utilizó la técnica denominada "*bagging*", la cual trata de separar los datos en múltiples cúmulos ("*bags*") con el fin de reducir la varianza (y el *overfitting*).

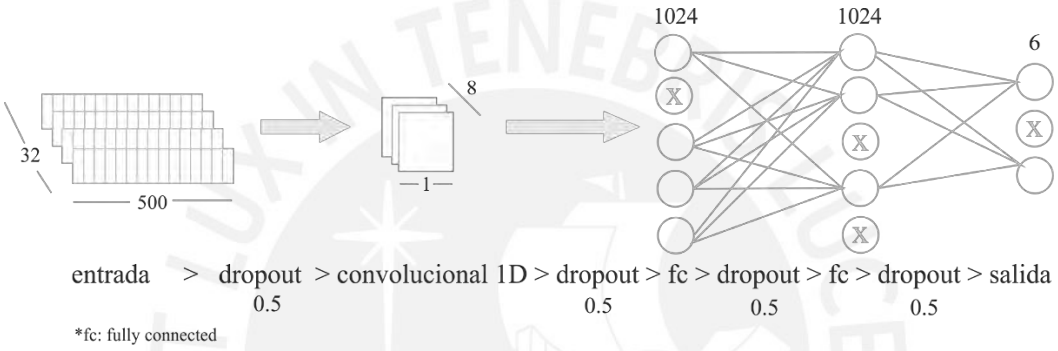


Figura 4.8. Arquitectura modelo CNN 1D. Fuente: Fuente propia

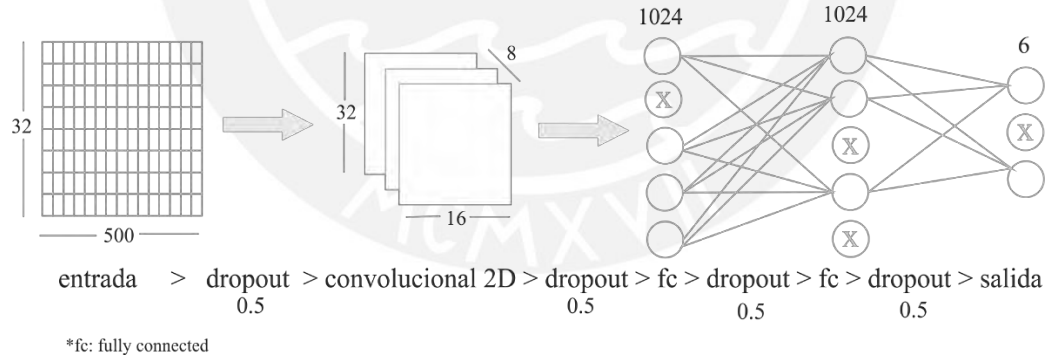


Figura 4.9. Arquitectura modelo CNN 2D. Fuente: Fuente propia

Tabla 4.4. Parámetros de los modelos tipo convolucionales. Fuente: Fuente propia

Modelo	Filtros de corte	Submuestreo	Bags
CNN 1D 5	5 Hz	8	10
CNN 1D 7-30	7 Hz, 30 Hz	8	10
CNN 1D 30 -1	30 Hz	8	10
CNN 1D 30 -2	30 Hz	4	10
CNN 2D 30	30 Hz	8	10

Modelos Redes Neuronales Recurrentes: El modelo de redes neuronales recurrentes utiliza una capa denominada *General Recurrent Unit* (GRU), utilizadas comúnmente para evitar el problema de la gradiente desvaneciente, un error común en RNN en el que el error propagado se hace tan pequeño que la red no puede memorizar datos ocurridos muy atrás en la secuencia. Después de esta capa, continúa una capa *fully connected* con 256 nodos. Igual que las redes convolucionales, se utiliza la técnica de "*dropout*". Asimismo, se realiza un procesamiento por filtro de bancos. En la Figura 4.10, se observa de manera gráfica la arquitectura mencionada y en la Tabla 4.5, se muestran los parámetros de procesamiento.

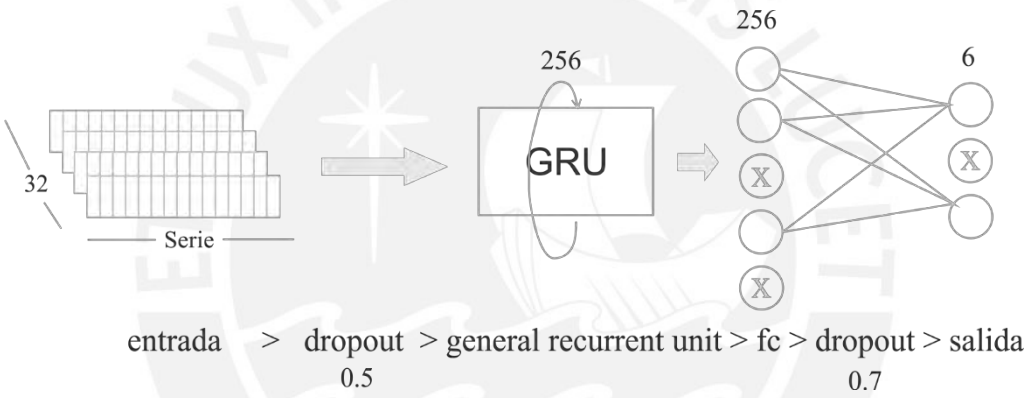


Figura 4.10. Arquitectura modelo RNN. Fuente: Fuente propia

Tabla 4.5. Parámetros de los modelos tipo recurrente. Fuente: Fuente propia

Modelo	Filtros de corte	Normalización
RNN	1Hz, 5 Hz, 10 Hz, 30 Hz	Estandarización

4.2.3. Pseudocódigo del programa

El programa consta de ocho archivos escritos en Python. El archivo con nombre `classify.py` es el programa principal, mientras que los otros siete son funciones que utiliza este programa. Tres son para clasificar modelos del nivel 1; dos, para el nivel 2; uno para el nivel 3; y el octavo archivo se encarga de enviar los eventos obtenidos al programa para comandar el brazo Kinova. Las entradas del programa algoritmo de traducción son el nombre del archivo

con las 400 muestras EEG y el número del sujeto. En la Figura 4.11, se muestra el diagrama flujo del programa `classify.py`. Debajo, se presentan los pseudocódigos de los seis archivos restantes.

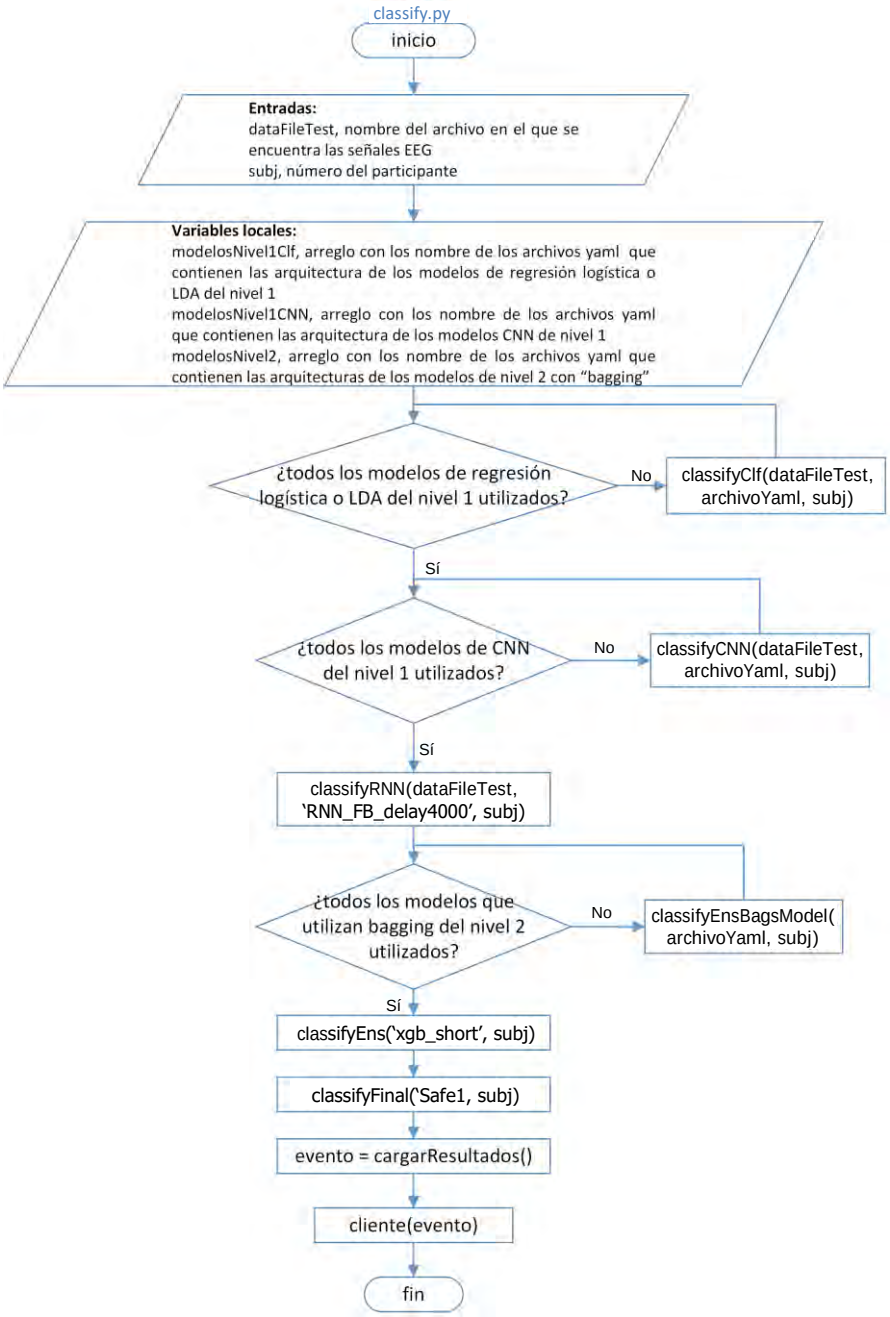


Figura 4.11. Diagrama de flujo del programa principal del Algoritmo de Traducción. Fuente: Fuente propia

`classifyClf.py`:

La función `classifyClf.py` se ejecuta de la siguiente manera: primero, extrae dos parámetros con nombres "`fileName`" y "`clfNum`". El primer parámetro es de tipo *string* e indica tanto el nombre del modelo como el procesamiento que se va a utilizar; mientras que el segundo parámetro indica mediante un número, el clasificador con el que se van a obtener las clasificaciones. Posteriormente, carga las señales EEG de las cuales se harán las clasificaciones. Seguidamente, procesamos los datos, luego de cargar los debidos modelos de pre y posprocesamiento. Con los datos procesados, carga los pesos y arquitecturas del clasificador que utilizará, para luego clasificar los resultados y guardarlos en un archivo .npy. El diagrama de pseudocódigo se muestra en la Figura 4.12.

```
function classifyClf:
    entradas: dataFileTest, nombre del archivo en el que se encuentran las señales EEG
              yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo
              subj, número del sujeto/participante

    variables locales: fileName, nombre del modelo
                       clfNum, número de clasificadores en el modelo
                       data, señales EEG cargadas del archivo dataFileTest
                       preprocessing, submodelo de preprocesamiento
                       postprocessing, submodelo de postprocesamiento
                       dataProcesada, señales EEG procesadas
                       clf, submodelo de clasificación
                       preds, matriz con las predicciones para cada columna (evento)
                       preds_tot, matriz con las predicciones totales

    [fileName, clfNum] ← extraerDatosYaml(yamlFile)
    data ← cargarSeñalesEEG(subj, dataFileTest)
    preprocessing ← cargarModeloPreprocesamiento(fileName, subj)
    dataProcesada ←preprocesarData(preprocessing,data)
    postprocessing ← cargarModeloPostprocesamiento(fileName, subj)
    dataProcesada ←postprocesarData(postprocessing,dataProcesada)
    para cada nClf en clfNum:
        para cada col del 0 al 5:
            clf ← cargarModelo(fileName, subj, nClf, col)
            preds.anexar(predecir(clf, dataProcesada))
        preds_tot←concatenarFilas(preds)
    guardar(preds_tot, fileName)
Fin de función
```

Figura 4.12. Pseudocódigo de la función `classifyClf.py`. Fuente: Fuente propia

`classifyCNN.py`:

La función `classifyCNN.py` se ejecuta de la siguiente manera: primero, se extraen tres parámetros del archivo yaml "`fileName`", "`bags`" y "`filtros`". A diferencia de la función anterior, el parámetro "`fileName`" no indica el procesamiento a utilizar; por el contrario, indica el nombre del archivo donde están guardado los pesos del modelo CNN. El procesamiento de los modelos CNN es solamente un banco de filtros, cuya cantidad está definida en el parámetro "`filtros`". En este caso, se utilizó la técnica de *bagging* en el entrenamiento; por ello, se realiza una clasificación para cada *bag*. Cuando se han realizado todas las clasificaciones, estas se guardan en un archivo con terminación .npy. El pseudocódigo del programa mencionado se muestra en la Figura 4.13.

```
function clasificarCNN:
   entradas: dataFileTest, nombre del archivo en el que se encuentran las señales EEG
             yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo
             subj, número del sujeto/participante

   variables locales: fileName, nombre del modelo
                      bags, número de bags (bolsas) del modelo
                      filtros, número de filtros del modelo
                      data, señales EEG cargadas del archivo dataFileTest
                      modelo, modelo de redes neuronales convolucionales
                      preds, matriz con las predicciones para cada bag
                      preds_tot, matriz con las predicciones totales

   [fileName, bags, filtros] ← extraerDatosYaml(yamlFile)
   para cada bag en bags:
       data ← cargarSeñalesEEGYProcesar(dataFileTest, subj, filtros)
       modelo ←crearCNN(data)
       modelo.cargarParametros(fileName, subj, bag)
       preds ← modelo.predecir()
       preds_tot.anexar(preds)
    preds_tot ← mediaPorFila(preds_tot)
    guardar(preds_tot, fileName)
Fin de función
```

Figura 4.13. Pseudocódigo de la función `classifyCNN.py`. Fuente: Fuente propia

`classifyRNN.py`:

La función `classifyRNN.py` se muestra en la Figura 4.14. La ejecución es similar a `classifyClf.py`. La diferencia se encuentra en que no hay múltiples clasificadores en un archivo yaml, y las clasificaciones no se realizan por columna.

function classifyRNN:
entradas: *dataFileTest,* nombre del archivo en el que se encuentran las señales EEG
yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo
subj, número del sujeto/participante

variables locales: *fileName,* nombre del modelo
data, señales EEG cargadas del archivo *dataFileTest*
modelo, modelo de redes neuronales recurrentes
preprocessing, submodelo de preprocesamiento
postprocessing, submodelo de postprocesamiento
dataProcesada, señales EEG procesadas
preds, matriz con las predicciones totales

fileName ← extraerDatoYaml(*yamlFile*)
data ← cargarSeñalesEEG(*dataFileTest*, *subj*)
preprocessing ← cargarModeloPreprocesamiento(*fileName, subj*)
postprocessing ← cargarModeloPostprocesamiento(*fileName, subj*)
modelo ← cargarModelo(*fileName*, *subj*)
dataProcesada ←preprocesarData(*preprocessing*, *data*)
dataProcesada ←postprocesarData(*postprocessing*, *dataProcesada*)
preds ← *modelo*.predecir(*dataProcesada*)
guadar(*preds, fileName*)
Fin de función

Figura 4.14. Pseudocódigo de la función `classifyRNN.py`. Fuente: Fuente propia

`classifyEns.py`:

La función `classifyEns.py` es una función del nivel 2. El pseudocódigo se muestra en la Figura 4.15, y la explicación se detalla a continuación. Primero, se extraen dos parámetros del archivo yaml "`fileName`" y "`ensemble`". El archivo "`fileName`" indica el nombre con el que se guardó el modelo de nivel 2. Por otro lado, el archivo "`ensemble`" indica los nombres de los

modelos de nivel 1, cuyas clasificaciones se cargarán como entradas para el presente modelo de nivel 2. Al terminar la clasificación, los resultados se guardan en un archivo .npy.

```
function classifyEns:
   entradas: yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo
             subj, número del sujeto/participante

   variables locales: fileName, nombre del modelo
                      ensemble, contiene los nombres de los modelos de nivel 1, cuyas
                          predicciones serán las entradas del correspondiente modelo de
                          nivel 2
                     data, predicciones de nivel 1 que son entradas para el presente
                          modelo
                     modelo, modelo basado en redes neuronales
                     preds, matriz con las predicciones

   [fileName, ensemble] ← extraerDatoYaml(yamlFile)
   data ← cargarPrediccionesLvl1(subj, ensemble)
   modelo ← cargarModelo(fileName)
   modelo.cargarPesos(fileName)
   preds ← modelo.predecir(data)
   guadar(preds, fileName)
Fin de función
```

Figura 4.15. Pseudocódigo de la función `classifyEns.py`. Fuente: Fuente propia

`classifyEnsBags.py`:

La función `classifyEns.py` es una función del nivel 2 similar a la anterior, cuyo pseudocódigo se muestra en la Figura 4.16. Igualmente, se extraen los parámetros "`fileName`" y "`ensemble`" como en la función anterior. No obstante, en esta función también se extra el parámetro "`bags`", puesto que estos modelos fueron entrenados con la técnica *bagging*. Al terminar la clasificación, los resultados se guardan en un archivo .npy.

`clasiifyFinal.py`:

La función `classifyFinal.py` es una función del nivel 3. De manera similar a las funciones de nivel 2, se extraen los parámetros "`fileName`" y "`ensemble`". En este caso, "`ensemble`" indica las clasificaciones de los modelos de nivel 2 a utilizar. Al terminar la clasificación, los

resultados se guardan en un archivo .npy. El pseudocódigo de la función detallada se muestra en la Figura 4.17.

`cliente.py`:

La función `cliente.py` se comunica con la función `servidor-kinova.py` (que se mostrará en la Sección 4.2.3) con el fin de transmitir el evento del programa algoritmo de traducción al programa comandar brazo Kinova. El pseudocódigo de la función se encuentra en la Figura 4.18.

```
function classifyEnsBags:
    entradas: yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo
              subj, número del sujeto/participante

    variables locales: fileName, nombre del modelo
                       ensemble, contiene los nombres de los modelos de nivel 1, cuyas
                       predicciones serán las entradas del correspondiente modelo de nivel 2
                       bags, número de bags (bolsas) del modelo
                       data, predicciones de nivel 1 que son entradas para el presente
                       modelo
                       modelo, modelo basado en redes neuronales
                       preds, matriz con las predicciones

    [fileName, ensemble, bags] ← extraerDatoYaml(yamlFile)
    data ← cargarPrediccionesLvl1(subj, ensemble)
    para cada bag in bags:
        modelo ← cargarModelo(fileName)
        modelo.cargarPesos(fileName)
        preds ← (preds + modelo.predecir(data)) / (bags)
    guadar(preds, fileName)
Fin de función
```

Figura 4.16. Pseudocódigo de la función `classifyEnsBags.py`. Fuente: Fuente propia

```
function classifyFinal:
   entradas: yamlFile, nombre del archivo yaml que detalla la arquitectura del modelo

   variables locales: fileName, nombre del modelo
                      ensemble, contiene los nombres de los modelos de nivel 2, cuyas
                       predicciones serán las entradas del correspondiente modelo de nivel 3
                      modelo, modelos de clasificación por media aritmética, geométrica y
                       de potencia data, predicciones de nivel 2 que son entradas para el
                       presente modelo
                      preds, matriz con las predicciones

   [fileName, ensemble] ← extraerDatoYaml(yamlFile)
   data ← cargarPrediccionesLvl2(subj, ensemble)
   para cada m en [0,1,2]:
       modelo ← cargarModelo(fileName)
       preds ← (preds + modelo.predecir(data)) / 3
   guadar(preds, fileName)
Fin de función
```

Figura 4.17. Pseudocódigo de la función `classifyFinal.py`. Fuente: Fuente propia

```
function cliente:
   entradas: data, variable String con la información a enviar

   variables locales: sock, permite el intercambio de datos por internet
                      server_address, dirección del servidor

   sock = socket.socket(AF_INET, SOCK_STREAM)
   sock.conectarAServidor(server_address)
   sock.enviar(data)
   sock.cerrar()
Fin de función
```

Figura 4.18. Pseudocódigo de la función `cliente.py`. Fuente: Fuente propia

4.2. Comando del brazo Kinova: Diseño

El brazo Kinova efectuará las acciones que indica el algoritmo de traducción. En esta sección, se describirán los programas a utilizar del paquete Kinova ROS, se determinará la posición

del efector final para cada una de las seis acciones y se presentará el pseudocódigo del programa.

4.2.1. Kinova ROS

Kinova Robotics ha desarrollado un paquete en ROS para la comunicación y el control con el brazo Kinova. Este paquete se utiliza en conjunto con la librería `actionlib`, la cual implementa acciones. Estas acciones funcionan igual a un cliente o un servidor. A continuación, en la Tabla 4.6, se describen los programas utilizados en el presente trabajo.

Tabla 4.6. Paquete ROS para el brazo Kinova. Fuente: Fuente propia.

Programa	Descripción
kinova_robot.launch	Ejecuta los drivers del brazo Kinova. Además, recibe como argumento un *string* de ocho caracteres para configurar el Kinova. El primer carácter se refiere al modelo del Kinova. Puede tomar los valores j para jaco, m para mico, r para roco y c para "*customized*" (personalizado). El siguiente carácter es la versión, que puede ser 1 o 2. Continúa con el tipo de muñeca: esférico (s) o no (n). Seguidamente se escribe el número de grados de libertad: 4, 6 o 7. El quinto carácter especifica si el robot es de servicio (s) o de asistencia (a). El número de dedos, dos o tres, en el gripper se detalla en el sexto carácter. Los dos últimos caracteres toman el valor de cero por default. Por ejemplo, para el presente trabajo, el valor *string* debe ser m2n6a200.
kinova_tool_pose_action.cpp	Este archivo contiene un servidor que recibe el punto cartesiano y la orientación (en cuaternión) al que debe moverse el efector final. El programa para comandar el brazo Kinova será el que envíe los datos mencionados como el cliente.
kinova_fingers_action.cpp	Este archivo contiene un servidor que recibe tres números entre 0 y 6400, en el que cero indica un dedo completamente abierto y 6400, completamente cerrado. Cuando se utiliza un *gripper* de dos dedos, el tercer dedo debe ser puesto a cero para evitar retrasos en los resultados. Igual que con el servidor anterior, el programa para comandar el brazo Kinova actuará como el cliente.

4.2.2. Descripción del brazo Kinova

La Figura 4.19 muestra las medidas del brazo Kinova MICO así como el origen de su sistema de referencia, el cual se encuentra en la base inferior. En la imagen referida, el eje Z se encuentra en la dirección de los eslabones (derecha), el eje Y en la dirección perpendicular hacia arriba, y el eje X en la dirección perpendicular a ambas.

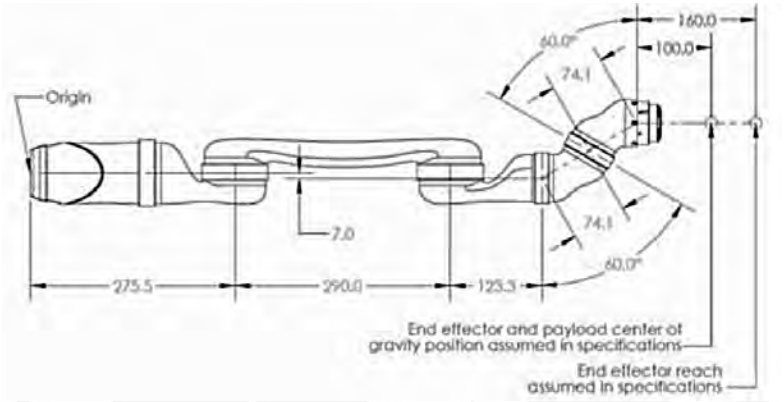


Figura 4.19. Dimensiones del brazo Kinova MICO. Fuente: [7]

A continuación, se muestra la Tabla 4.7, la cual está dividida en seis secciones que corresponden a cada evento que puede ser interpretada por el algoritmo de traducción. En cada sección, se muestran las coordenadas del efector final (x, y, z) y el estado del *gripper*, así como una imagen de referencia, en la que se muestra el brazo, las distancias y el objeto; para el caso presente, una botella.

Para todas las posiciones, la orientación del *gripper* siempre es la misma. Estos ángulos son {0.785, 1.57, 0.785} radianes en el sistema XYZ de Euler. Sin embargo, el paquete Kinova ROS recibe la orientación representados con un cuaternión. A continuación, se hallará la representación en cuaternión de los ángulos en cuestión.

Sean:

$$\theta_x = 0.785\ rad, \theta_y = 1.57\ rad, \theta_z = 0.785\ rad.$$

También se definen:

$$sx = \sin(0.5 * \theta_x),$$

$$cx = \cos(0.5 * \theta_x),$$

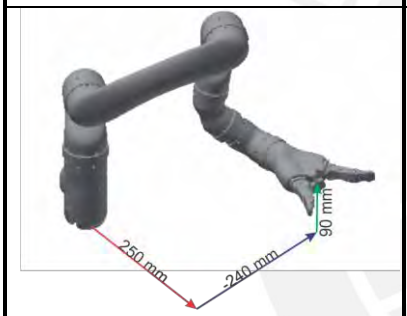
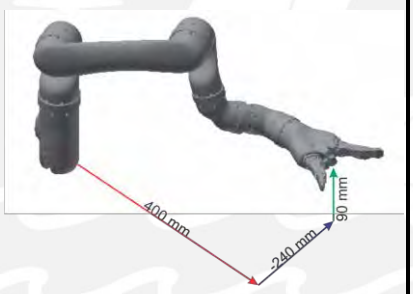
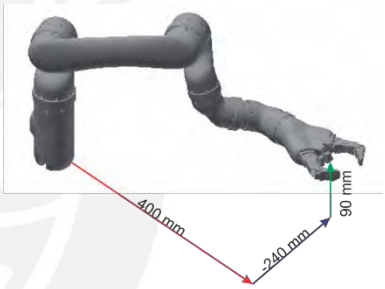
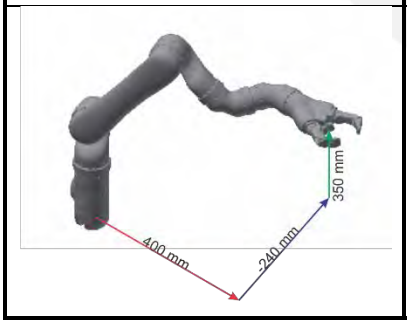
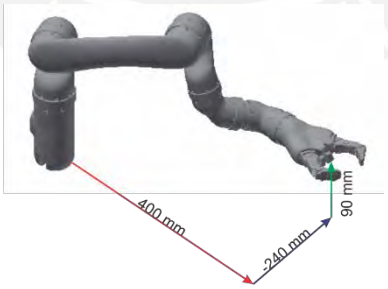
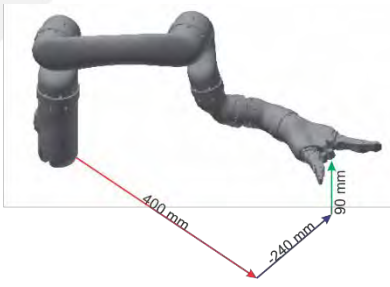
$$sy = \sin(0.5 * \theta_y),$$

$$cy = \cos(0.5 * \theta_y),$$

$$sz = \sin(0.5 * \theta_z),$$

$$cz = \cos(0.5 * \theta_z).$$

Tabla 4.7. Posiciones posibles para el brazo Kinova*. Fuente: Imágenes obtenidas del modelo en inventor ofrecido por *Kinova Robotics*.

Inicio del movimiento	Toque con los dedos	Aplicación de fuerza
(250, -240, 90) – abierto	(400, -240, 90) – abierto	(400, -240, 90) – cerrado
		
Levantamiento	**Reubicación**	**Soltar objeto**
(400, -240, 350) – cerrado	(400, -240, 90) – cerrado	(400, -240, 90) – abierto
		

* Una posición que no se muestra es la posición inicial (“home”) del brazo, cuyas coordenadas son (210,-260,480).

Luego, las ecuaciones de transformación son las siguientes:

$$q_0 = sx * cy * cz + cx * sy * sz,$$

$$q_1 = -sx * cy * sz + cx * sy * cz,$$

$$q_2 = sx * sy * cz + cx * cy * sz,$$

$$q_3 = -sx * sy * sz + cx * cy * cz.$$

Finalmente, la representación en cuaternión:

$$Q = \{q_0, q_1, q_2, q_3\} = \{0.4998, 0.4999, 0.4998, 0.5005\}.$$

4.2.3. Pseudocódigo del programa

Un nodo en ROS describe un proceso que efectúa el robot. El presente trabajo implementa dos nodos: "`comandar Kinova`" y "`servidor Kinova`". La comunicación en ROS se da mediante dos formas. En la primera, los nodos toman un rol, el de suscriptor o el de publicador. Los nodos publicadores escriben un mensaje a un tópico; los nodos que estén suscritos a este tópico ejecutan una función cada vez que reciben el mensaje. La segunda forma permite a los nodos tomar el rol de servidor o cliente. El nodo cliente solicita una conexión con el nodo servidor. El nodo servidor acepta la conexión y el nodo cliente envía su mensaje.

El programa `servidor-Kinova` es un nodo publicador. Este programa recibe la acción a realizar a través de una conexión directa a internet. Seguidamente, envía el mensaje con la acción al nodo suscriptor, que en este caso es el programa comandar-brazo. Al recibirlo, se ejecuta la función `interpretarComando`. Esta función asigna la posición del efector final y la apertura de los dedos a dos mensajes. Posteriormente, el nodo comandar brazo Kinova, que también es un nodo cliente, envía los mensajes a los nodos servidores "`kinova_tool_pose_action`" y "`kinova_fingers_action`", que se encargan de enviar los ángulos de referencia a los motores del Kinova. En la Figura 4.20, se muestra el diagrama de nodos, donde los nodos de abajo son los nodos implementados y los de arriba, son nodos pertenecientes al paquete Kinova-ROS. Asimismo, se muestran los pseudocódigos de los programas por implementar y sus funciones.

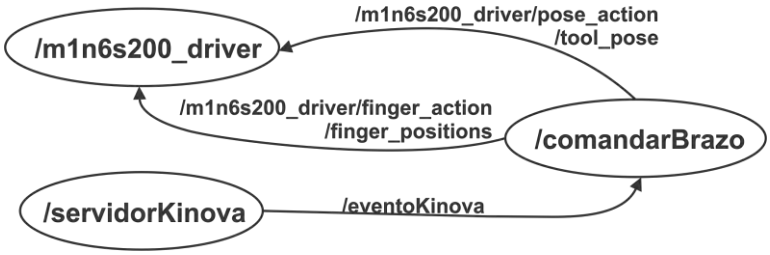


Figura 4.20. Diagrama de nodos. Fuente: Fuente propia

servidor-Kinova.py:

Este programa se desarrollará en python y tendrá la función de esperar un mensaje con el evento que el brazo Kinova debe efectuar. Cuando recibe el evento, lo publica inmediatamente en el tópico `/eventoKinova` para que pueda ser leído por el programa `comandar-brazo`. El pseudocódigo de este programa se muestra en la Figura 4.21.

```
programa servidor-Kinova
   variables locales: pub, nodo publicador de ROS
                      sock, permiten el intercambio de datos por internet
                      connection, conexión con un cliente
                      data, data recibida
   pub ← rospy.Publisher()
   rospy.iniciar_nodo()

   sock ← socket.socket(AF_INET, SOCK_STREAM)
   sock.bind()

   while 1:
       connection ← sock.aceptar_conexion()
       data ← connection.recibir()
       if data:
           pub.publicar(data)
Fin de programa
```

Figura 4.21. Pseudocódigo del programa `servidor-Kinova.py`. Fuente: Fuente propia

comandar-brazo.cpp:

Este programa, el cual se muestra en la Figura 4.22, se desarrollará en c++ y tiene la función de enviar la posición de referencia y estado del *gripper* a los programas `kinova_tool_pose_action.cpp` y `kinova_fingers_action.cpp` del paquete ROS. Este programa llama a la función `interpretarComando` cuando recibe el evento que debe ejecutar el brazo.

4.3. Integración de los módulos: Diseño

En la presente sección, se definen los parámetros de diseño referidos a la integración del sistema: la selección de componentes, la comunicación entre los módulos, la alimentación y el consumo de energía del sistema.

4.3.1. Selección de componentes

El sistema funciona con cuatro componentes físicos los cuales son un sensor para la medición de señales electroencefalográficas, un brazo robot y dos computadoras portátiles para procesar los algoritmos.

En la Tabla 4.8, se muestran los componentes seleccionados y las razones por la que es apto para el sistema.

4.3.2. Comunicación entre módulos

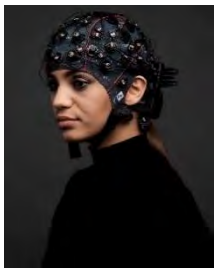
El trabajo presente se compone de tres sistemas de comunicación entre los siguientes cuatro componentes: el sensor EEG g.Nautilus, las dos computadoras portátiles para el procesamiento de los algoritmos de traducción (laptop 1) y para comandar el brazo Kinova (laptop 2), y el brazo Kinova. En la Figura 4.26, se muestra el esquema de comunicaciones y, en la Tabla 4.9, se detallan por escrito las características de cada uno. En el caso de la primera y tercera columna, la frecuencia, el canal y el protocolo están definidos en las guías

del g.Nautilus [28] y el brazo Kinova [29]. Se ha escogido comunicación por medio de WiFi para ambas laptops para evitar el uso de cables que limitan la separación entre el usuario y el brazo Kinova, así como también para garantizar la compatibilidad con los trabajos previos en el tema de BCI realizados en la PUCP.

```
variables globales: posiciones, un arreglo de dos dimensiones de 3x6 elementos con las
                        posiciones del efector final para cada evento
                    poseDedos, un arreglo de dos dimensiones de 6x2 elementos que
                        indican la apertura de cada dedo del gripper
                    orientaciones, un arreglo de dos dimensiones de 6x4 elementos con las
                        orientaciones del efector final para cada evento

function interpretarComando(msg)
   entradas: msg, mensaje que contiene el evento a realizar
   variables locales: evento, acción a efectuar
                      cliente1, cliente para el servidor que controla la posición del efector
                          final
                      cliente2, cliente para el servidor que controla el gripper
                      meta1, contiene la posición del efector final
                      meta2, indica la apertura de cada dedo
   evento<-msg.data
   meta1.posicion.x ← posiciones[evento][0]
   meta1.posicion.y ← posiciones[evento][1]
   meta1.posicion.z ← posiciones[evento][2]
   meta1.orientacion.x ← orientaciones [evento][0]
   meta1. orientacion.y ← orientaciones [evento][1]
   meta1. orientacion.z ← orientaciones [evento][2]
   meta1. orientacion.w ← orientaciones [evento][3]
   meta2.finger1 ← poseDedos[evento][0]
   meta2.finger2 ← poseDedos[evento][1]
   meta2.finger3 ← 0

   cliente1.enviarMensaje(meta1)
   cliente1.esperarResultado()
   cliente2.enviarMensaje(meta2)
   cliente2.esperarResultado()
Fin de función

programa comandar-brazo
   variables locales: nh, nodo de ROS

   iniciarROS("comandarBrazo")
   nh.subscribe("comandarBrazo", interpretar_comando)
   ros::spin()
Fin de programa
```

Figura 4.22. Pseudocódigo del programa `comandar-brazo.cpp`. Fuente: Fuente propia

Tabla 4.8. Selección de componentes. Fuente: Fuente propia

Componente	Razón para la selección
g.Nautilus  Figura 4.23. g.Nautilus. Fuente: [30]	El sensor EEG adecuado debe tener las mismas características que el sensor EEG utilizado para la adquisición de señales en la base de datos WAY-EEG-GAL. Estas características son su ratio de muestreo de 500 Hz y contar con 32 canales. El g.Nautilus es una solución de mediano costo con precisión aceptable que cumple con estos requisitos.
Laptop  Figura 4.24 Laptop. Fuente: [31]	Ambos programas, algoritmo de traducción y comando de Kinova, pueden funcionar con cualquier tipo de computadora portátil comercial moderna (como la de la Figura 4.24). Sin embargo, para un desempeño aceptable, el algoritmo de traducción necesita de una computadora portátil con siete núcleos, tarjeta gráfica NVidia y 16 GB de memoria RAM.
Brazo Kinova MICO  Figura 4.25. Brazo Kinova. Fuente: [32]	Puesto que el brazo robot debe trabajar con personas en su día a día, este debe ser del tipo de servicio o asistencial. El brazo robot Kinova MICO ha sido diseñado exactamente con el propósito de asistir a personas con discapacidades motoras. Su baja velocidad de traslación y sus sensores ayudan a prevenir lastimar a alguna persona cerca de este.

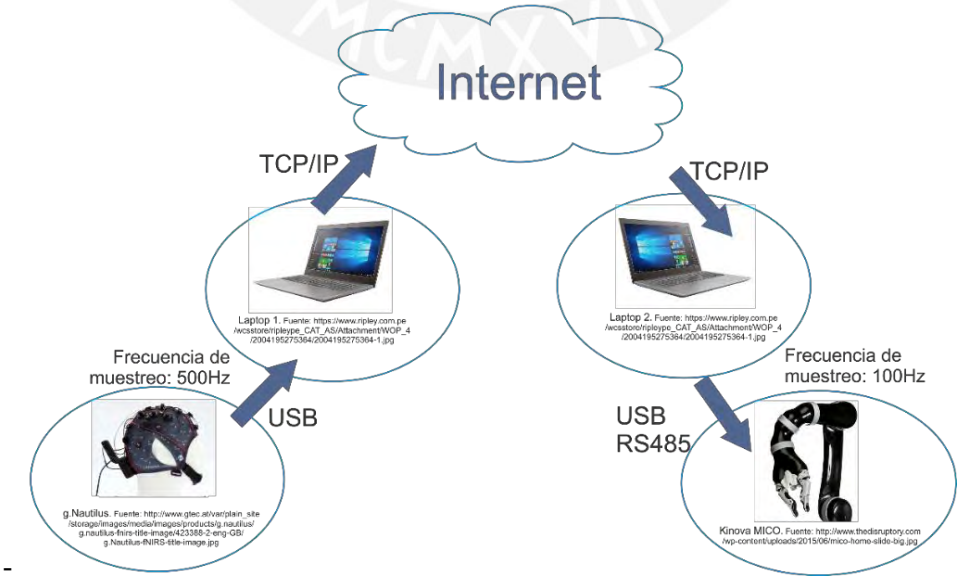


Figura 4.26. Diagrama de comunicaciones. Fuente: Fuente propia

Tabla 4.9. Descripción de los medios de comunicación. Fuente: Fuente propia.

Sistema de Comunicación	g.Nautilus/Laptop 1	Laptop 1/Laptop 2	Laptop 2/ Kinova
Frecuencia de transmisión de datos	500 Hz	Dependiente del servidor	100 Hz
Canal	USB	Wifi	USB
Protocolo de Red	USB	IPv6	USB

4.3.3. Consumo de energía

Cuatro componentes consumen energía en el sistema. El primero, el g.Nautilus, tiene una batería de ION Litio que dura 8 horas continuas y necesita 3 horas para recargarse completamente [28]. También sigue la norma PI 967 IATA [33], esta requiere que las baterías entreguen como máximo 100 Wh. Conociendo esto, se puede hallar la energía que consume el dispositivo en una hora.

$$\text{Consumo} = \frac{100Wh}{8h} = 12.5W,$$

$$\text{Consumo x Hora}_{Nautilus} = 10W * 1h = 12.5Wh.$$

Un análisis detallado del consumo de energía en laptops, se puede encontrar en [34]. El autor del informe señala que una laptop consume entre 20W a 50W. Para fines de este trabajo, se tomará 50W como el consumo de energía de cada laptop.

$$\text{Consumo x Hora}_{Laptop} = 50W * 1h = 50Wh.$$

La guía de usuario del brazo Kinova MICO indica que la potencia que consume el robot en modo de operación es de 25W [7]. Por consiguiente, el consumo por hora es de 25Wh.

$$\text{Consumo x Hora}_{Kinova} = 25\ W * 1\ h = 25Wh.$$

En la Tabla 4.10, se muestra el consumo de energía de cada dispositivo por hora y el total.

Tabla 4.10. Consumo de energía en una hora. Fuente: Fuente propia

Componente	Energía (Wh)
g.Nautilus	12.5
Laptop 1	40
Laptop 2	40
Brazo Kinova	25
Total	117.5

4.4. Protocolo del experimento con el usuario: Diseño

En la sección 4.1, se mencionó que las señales con las que se está entrenando el algoritmo de traducción son parte de un *dataset*. Sin embargo, clasificar las activaciones de las funciones de agarre y levante a partir de señales que provienen de un *dataset* es muy distinto a clasificarlas con señales adquiridas en tiempo real de participantes. Esto último se asemeja más a cómo se utilizarían los dispositivos en un caso real. Por ello, este trabajo propone el diseño de un experimento con el usuario y toma como guías los trabajos "*Practical guide to performing an EEG experiment*" [35] y "*Write a protocol*" [36], así como el experimento referido para obtener el *dataset* [5].

4.4.1. Planeamiento del experimento

El propósito de esta sección es explicar los requerimientos para participar en el proyecto, la organización de los tiempos en el experimento y, finalmente, el componente ético.

Selección de participantes:

Un participante del experimento debe cumplir con lo siguiente: ser mayor de 18 años, ser diestro y no sufrir de alguna enfermedad física o mental. Además, se pedirá a los

participantes encontrarse en las mejores condiciones físicas y mentales durante el experimento, puesto que ello influye sustancialmente en los resultados.

Organización de tiempos:

El experimento contará con cinco sesiones de 60 minutos cada una. Cada sesión cuenta con 3 pruebas, cuya tarea a realizar se describe en el punto 4.4.3 y su duración es de aproximadamente 10 minutos. Dado que la tarea puede causar estrés físico y mental en el participante, se darán descansos de 5 minutos. Este tiempo se podrá extender si el participante lo requiere, con el fin de asegurar su bienestar.

Antes de empezar las pruebas, se le dará a cada participante una copia con las instrucciones de la tarea a realizar. Asimismo, se hará una prueba de entrenamiento con duración de 5 minutos, con el fin de que el participante pueda entender mejor la tarea.

Cumplimiento de la ética en la investigación:

A los participantes del experimento se les entregará el consentimiento informado y un documento con los detalles del experimento, de manera que puedan tomar una decisión consciente sobre su participación. Además, tendrán la posibilidad de cambiar de opinión y abandonar el experimento en cualquier momento, sin consecuencias negativas para él o ella. Este derecho también se les será informado. No se tiene previsto guardar los datos de las señales EEG de los usuarios. En caso esto cambie durante el desarrollo del trabajo, los datos obtenidos se utilizarán solamente para el presente estudio y se mantendrá la anonimidad de los participantes. Sí se guardarán las medidas de desempeño del algoritmo de traducción para cada uno de los participantes con el fin de evaluar el sistema. Estos datos también mantendrán la anonimidad. Asimismo, el participante tendrá conocimiento de todo lo referido anteriormente.

4.4.2. Protocolo experimental

Propósito: Evaluar el desempeño de las funciones de agarre y levante en exactitud (clasificaciones correctas sobre el total de clasificaciones) y tiempo que toma el participante en movilizar el brazo Kinova.

Materiales: Los equipos a utilizar para este experimento son los siguientes.

- g.Nautilus de la empresa g.tec con 32 canales y electrodos secos.
- g.USBamp de la empresa g.tec
- electrodos de referencia
- brazo Kinova MICO
- 1 computadora portátil con Windows 10, de preferencia con 7 núcleos y el software MATLAB
- 1 computadora portátil con Linux Ubuntu y el software ROS

Métodos:

- Una hora antes de la llegada del participante, se debe contar con los equipos y materiales preparados.
- Se acomoda al participante en posición de descanso sobre una silla.
- Se pone el casco EEG sobre la cabeza del participante con los electrodos como se muestra en la Figura 4.27. Asimismo, se comprueba que los electrodos hagan contacto con la piel.
- Una luz roja le indica el inicio de una prueba (detalles en la Sección 4.4.3).
- Se mide el tiempo que toma cada paso en la prueba
- Terminada la prueba, el participante regresa a la posición de descanso, mientras el investigador anota resultados y observaciones.

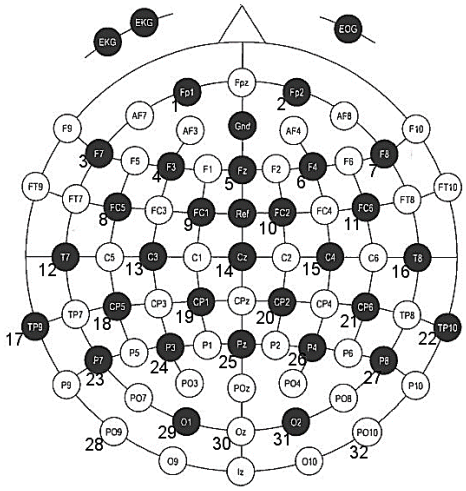


Figura 4.27. Posición de los electrodos en la cabeza. Fuente: [37]

Interpretación de datos: En este experimento, se van a tomar dos tipos de datos. El primero indica si para cada movimiento del participante, el brazo Kinova efectuó el movimiento correspondiente. A partir de este dato, se calculará la exactitud para cada evento. El segundo es el tiempo que toma el brazo Kinova en moverse luego de que el participante haya realizado un movimiento del brazo. Finalmente, se analizarán los datos y se comparará el sistema con otros similares mencionados en el estado del arte.

Descripción de la tarea: El participante comienza con su brazo en posición de reposo con su muñeca apoyado sobre un cojín en la mesa como se observa en la Figura 4.28. Una luz roja indicará que la tarea ha empezado. El primer paso consiste en acercar su mano de manera que envuelva el objeto sin tocarlo. En el siguiente paso, la mano del participante debe tocar el objeto sin aplicar fuerzas. En el tercer paso, el participante debe aplicar fuerza para levantar el objeto. Seguidamente, se levanta el objeto a la altura del mentón. Posteriormente, el participante debe regresar el objeto a su posición original. Para finalizar, el participante debe soltar el objeto. La prueba acaba allí, ahora el participante debe regresar su brazo a la posición de reposo.

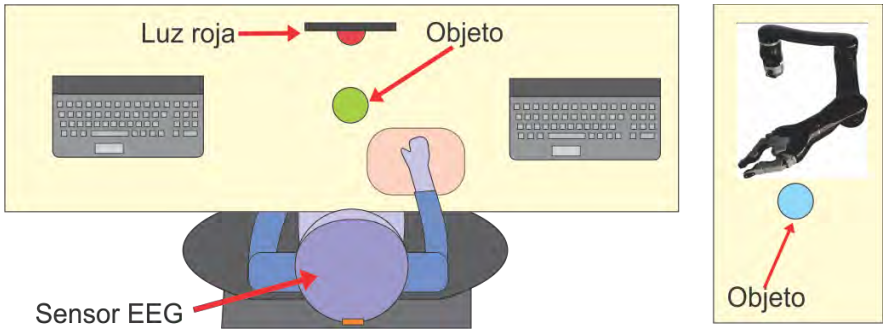


Figura 4.28. Disposición de los elementos en el experimento. Fuente: Fuente propia (brazo Kinova obtenido de [38])

Capítulo 5: PROYECTO DEFINITIVO

El presente capítulo presenta la codificación y posterior simulación de cuatro secciones del capítulo previo: el algoritmo de traducción y el comando del brazo Kinova.

5.1. Algoritmo de traducción: Programa y validación virtual

En esta sección se especifican las librerías del programa Algoritmo de Traducción. Este programa es similar al del capítulo anterior y, adicionalmente, se le agrega una función cliente que es parte de la comunicación entre módulos también mencionada en el capítulo previo. Finalmente, se muestran los resultados de una validación realizada con los datos de prueba, con el fin de verificar la métrica AUROC y el tiempo que toma el programa.

5.1.1. Programa del algoritmo de traducción

El programa Algoritmo-de-Traducción hace uso de ocho archivos, los cuales se pueden ver en el Anexo A. A continuación, se muestra la Tabla 5.1, donde se muestran la versión de Python y las librerías utilizadas.

Tabla 5.1. Versión de Python y librerías. Fuente: Fuente propia

Librerías		
Numpy 1.14.3	mne 0.10.0	Keras 0.1.2
Scipy 0.16.0	XGBoost 0.4a29	Lasagne 0.2.dev1
scikit-learn 0.17.1	Theano 0.8.0	Nolearn 0.6.0
pyriemann 0.2.3	hyperopt 0.0.2	pygpu 0.7.6
Versión de Python: 2.7		

5.1.2. Validación con datos de prueba

En [39], se comparan tres algoritmos también entrenados con el *dataset* denominado WAY-EEG-GAL, uno de los cuales está basado en una red neuronal convolucional, la cual utilizaremos como *benchmark*. Este modelo tomó 60 minutos en clasificar 3’144000 muestras, equivalente a 104.8 minutos de adquisición de señales EEG; en otras palabras, clasifica 52400 muestras/min. Asimismo, el desempeño de este modelo en la métrica AUROC fue de 0.829. La validación del modelo entrenado para este proyecto se realizó en una laptop de cuatro núcleos velocidad base 2.30 GHz con 7.81 GB de memoria RAM. Este modelo tomó 40 minutos en clasificar 30000 muestras, equivalente a 1 minuto de adquisición; es decir, el modelo clasifica 750 muestras/min. Además, este modelo obtuvo un desempeño en la métrica AUROC de 0.97. Se observa que hay una diferencia notable de tiempo entre ambos modelos. Una de las razones es que el modelo propuesto por este proyecto es más complejo y, por tanto, de mayor costo computacional.

En una aplicación real, tomar muestras durante un minuto, para posteriormente, recién poder brindar una clasificación no es una solución óptima. Tampoco lo es precargar los modelos convolucionales y recurrentes cada vez que se requiera realizar una clasificación; puesto que este paso se puede efectuar una sola vez, y, así es posible disminuir el tiempo de clasificación total. Tomando en cuenta lo mencionado, se realizó una prueba de validación adicional, considerando 500 muestras, equivalente a 1 segundo de adquisición. En esta prueba, el modelo tomó 38.6 segundos en brindar una clasificación. Este tiempo de clasificación, así como el tiempo para la prueba anterior, se muestran separados por nivel en la Tabla 5.2.

Tabla. 5.2. Resultados de la validación con datos de prueba. Fuente: Fuente propia

Modelos	Tiempo de clasificación de 30000 muestras (en segundos)	Tiempo de clasificación de 500 muestras (en segundos)
Nivel 1 (incluye preprocesamiento)		
Cargando a memoria modelos basados en redes neuronales convolucionales y recurrentes	105.39 s	100.32 s
Modelos basados en aprendizaje de máquinas	1793.87 s	30.47 s
Modelos basados en redes neuronales convolucionales	397.67 s	6.39 s
Modelo basado en redes neuronales recurrentes	57.35 s	0.79 s
Nivel 2		
Modelos basados en máquinas de aumento de gradiente (*XGBoost*)	47.68 s	0.942 s
Nivel 3		
Modelo de medias ponderadas	0.8190 s	0.014 s
Total		
Tiempo Total	2402.77 s	138.93
Tiempo Total (sin considerar la primera carga de los modelos)	2297.38 s	38.61 s
AUROC	0.97	

5.2. Comando del brazo Kinova: Programa y validación experimental

Esta sección se muestra el programa Comandar-Kinova. Como en la sección anterior, los programas desarrollados se encuentran en el Anexo al final del documento. Posteriormente, se muestran los resultados de una simulación, cuyas entradas están dadas por los seis eventos mencionados enviados uno después del otro.

5.2.1. Programa para comandar el brazo Kinova

El programa para comandar el brazo Kinova tiene dos archivos: `servidor-kinova`, escrito en Python y `comandarBrazo`, escrito en C++. Ambos archivos se encuentran en el Anexo B.

5.2.2. Validación experimental

Para la validación experimental del movimiento del brazo Kinova, se enviaron los seis eventos (mencionados en la Sección 4.1.1), uno seguido del otro, al programa `servidor-kinova`. Se hizo uso del programa *rqt* que pertenece a las librerías ROS, y se generaron tres gráficos, a partir de señales que provienen de los sensores del brazo. Cada gráfica pertenece a una validación distinta. En las tres, el brazo inició su movimiento desde su posición inicial ("*home*"). Asimismo, se muestran en líneas de color verde, el momento en que el efector final llega a la posición comandada para cada evento. A continuación, la Tabla 5.3 muestra los nombres y posiciones finales de cada evento.

Tabla 5.3. Información de cada evento. Fuente: Fuente propia.

Número del evento	Nombre del evento	Posición final (en m)	Estado del *gripper*
0	Inicio del movimiento	(0.25,-0.24,0.09)	Abierto
1	Toque con los dedos	(0.4,-0.24,0.09)	Abierto
2	Aplicación de fuerza	(0.4,-0.24,0.09)	Cerrado
3	Levantamiento	(0.4,-0.24,0.35)	Cerrado
4	Reubicación	(0.4,-0.24,0.09)	Cerrado
5	Soltar objeto	(0.4,-0.24,0.09)	Abierto

En la Figura 5.1, se muestran las posiciones en X, Y y Z del efector final en función del tiempo y en la Figura 5.2, se muestra la trayectoria espacial. Asimismo, en la Figura 5.3, se pueden ver las direcciones de los ejes X, Y y Z del robot Kinova y sus correspondientes articulaciones. Asimismo, se muestran las articulaciones en el brazo y el número que corresponden en la leyenda del gráfico. Se observa que, si bien el brazo llega a la posición deseada, el tiempo que toma para estabilizarse es relativamente largo. Afortunadamente, aunque a costa de reducir la precisión en la posición del efector final, se puede disminuir este tiempo mediante la modificación del programa.

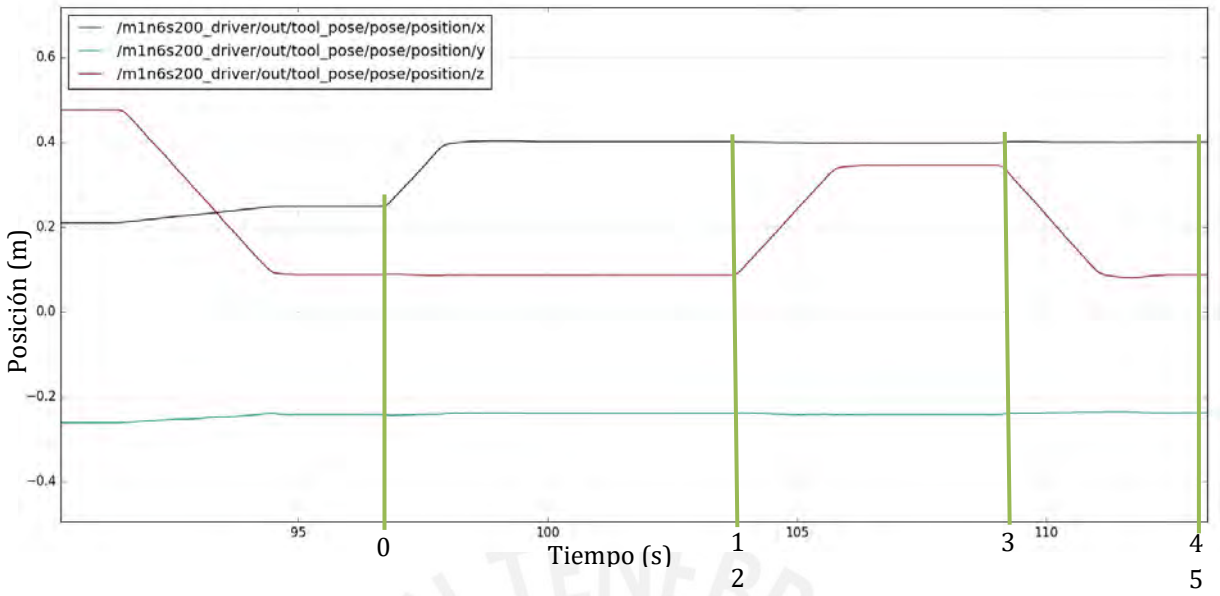


Figura 5.1. Posiciones del efector final en el tiempo. Fuente: Fuente propia

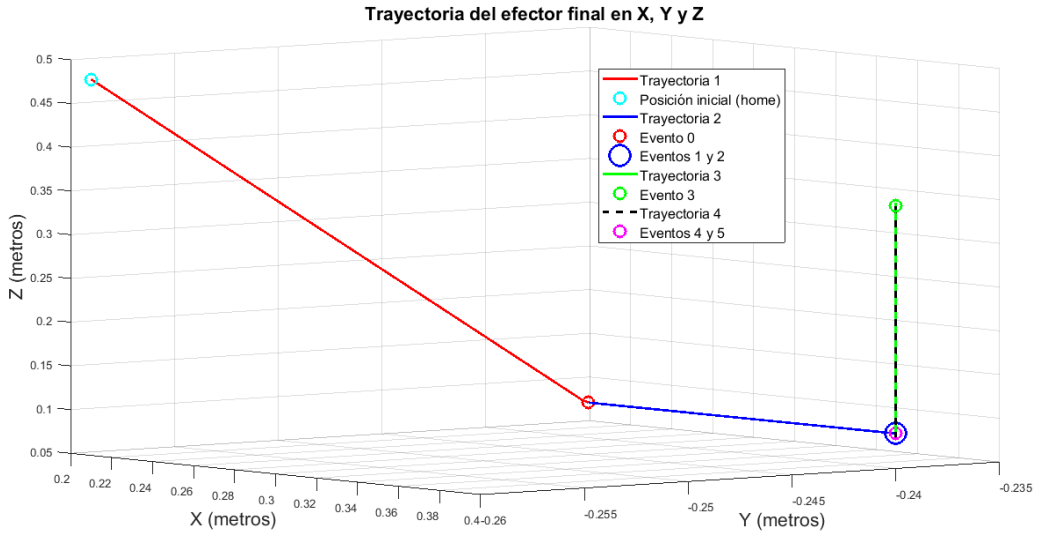


Figura 5.2. Trayectoria del efector final. Fuente: Fuente propia

Figura 5.3. Detalles Kinova MICO. Fuente: [40]

La Figura 5.4 muestra las posiciones angulares de las articulaciones, cada cual indicado en función a la numeración de la Figura 5.3. Se observa también que, así como las posiciones del efector final, el cambio de posición angular es aproximadamente lineal, después de llegar a la posición 0.

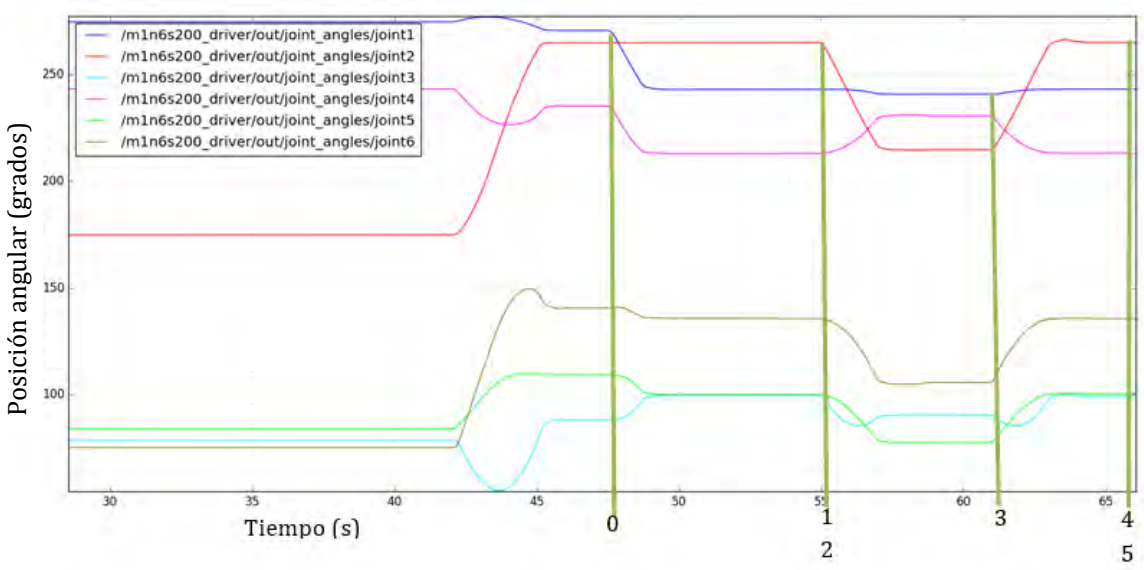
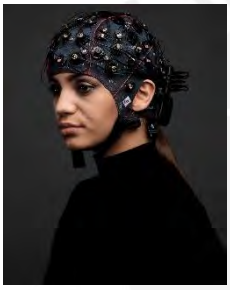


Figura 5.4. Posiciones angulares de las articulaciones en el tiempo. Fuente: Fuente propia

Capítulo 6: COSTOS

En el presente capítulo se detalla la lista de precios de los componentes físicos, los costos de diseño, de la integración y de las pruebas. A continuación, se presentan las Tabla 6.1, que precisa el precio de los componentes físicos, y la Tabla 6.2, que precisa las horas estimadas del diseño de ingeniería. Se observa que el costo total del sistema depende principalmente del precio de su componente más caro: el brazo Kinova.

Tabla 6.1. Lista de costo de los componentes. Fuente: Fuente propia

Componente	Costo
g.Nautilus  g.Nautilus. Fuente: [28]	$5,300 [28]
Laptop 1  Laptop 1. Fuente: [29]	$1350 [29]
Laptop 2  Laptop 2. Fuente: [29]	$1350 [29]

Brazo Kinova MICO  Brazo Kinova. Fuente: [30]	$25,850 [41]
Total	$33,850

Tabla 6.2. Estimación de horas de trabajo. Fuente: Fuente propia

Actividad		Horas
Diseño	Seleccionar los datos de entrenamiento	24
	Diseñar el algoritmo para entrenar el modelo	200
	Diseñar el algoritmo de traducción	120
	Diseñar la trayectoria del efector final del brazo robot	40
	Diseñar el algoritmo de comando de brazo robot	120
	Seleccionar los componentes físicos	32
	Diseñar la comunicación entre los programas desarrollados	48
	Documentar el diseño y el plan de implementación	80
Implementación	Construir y ejecutar el programa para el entrenamiento del modelo	400
	Construir y probar el algoritmo de traducción	200
	Construir y probar el algoritmo de comando de brazo robot	200
	Integrar y probar los programas desarrollados	100
	Validar los resultados e implementar las correcciones correspondientes	200
Horas totales		1764
Costo total (aprox. 4 meses de trabajo)		$ 2614.29

Asumiendo que el sueldo de un egresado de ingeniería mecatrónica es de S/. 2440 al mes [42], y siendo el costo del dólar S/. 3.5, el costo total resulta $2614.29. Agregado al costo de componentes, el costo total resulta $36464.

Si bien es cierto que el precio es elevado, los brazos robot tienden a durar más de 10 años, lo que lo convierte en una inversión a largo plazo. Además, existe la posibilidad de utilizar el

mismo algoritmo de traducción con un hardware distinto. Con el tiempo, el costo de los equipos irá disminuyendo sin reducir el desempeño; por consiguiente, cada vez más personas podrán favorecerse de esta solución. Cabe resaltar que la PUCP ya cuenta con los equipos mencionados, los cuales fueron utilizados en la elaboración de este trabajo.

CONCLUSIONES Y RECOMENDACIONES

En este capítulo, se describen las conclusiones que se han obtenido al finalizar el presente trabajo. Seguidamente, se presentan recomendaciones para un trabajo futuro.

Conclusiones:

- Se logró el objetivo de diseñar e implementar las funciones de agarre y levante en el brazo Kinova MICO mediante activación EEG, para esto se ha utilizado un programa denominado algoritmo de traducción, el cual demuestra un desempeño óptimo en la métrica AUROC, aunque su tiempo de clasificación no pudo superar el *benchmark*. Adicionalmente, con el programa `comandar brazo Kinova`, se logró llegar a todas las posiciones deseadas del brazo sin ningún inconveniente.
- Finalmente, se cumplieron con los requerimientos establecidos en la fase de diseño a excepción del requerimiento de control (deseo) que establece un tiempo de respuesta del brazo menor o igual a 5.0 ± 0.6s.

Recomendaciones:

- Es recomendable la ejecución de una prueba de laboratorio con el fin de determinar si el desempeño óptimo en la métrica AUROC equivale también a un desempeño óptimo en una situación real.
- Es seguro que se pueda disminuir el tiempo de clasificación si se utiliza una menor cantidad de modelos; sin embargo, también se podría reducir el desempeño en su capacidad de clasificación.
- Como trabajo futuro, se recomienda crear, con el fin de encontrar un punto medio entre tiempo de clasificación y capacidad de clasificación, los dos modelos siguientes: el primer modelo utiliza solamente métodos tradicionales de *machine learning* en su

primera capa (se obtiene una clasificación más veloz); el segundo modelo, solamente utiliza redes neuronales convolucionales (resultados más exactos). Ambos seguidos con una segunda capa compuesta de algoritmos "*xgboost*" y una tercera con medias ponderadas.

- El brazo Kinova MICO ha demostrado un tiempo de estabilización bajo; sin embargo, ha sido capaz de llegar a la posición final requerida. Si bien esto no ha afectado el trabajo presente, es importante tomar en cuenta en trabajos futuros en los que se requiera tiempos de reacción menores.

BIBLIOGRAFÍA

[1] L. R. Hochberg *et al.*, "Reach and grasp by people with tetraplegia using a neurally controlled robotic arm", *Nature*, vol. 485, nº 7398, pp. 372–375, 2012.

[2] J. Meng, S. Zhang, A. Bekyo, J. Olsoe, B. Baxter, y B. He, "Noninvasive Electroencephalogram Based Control of a Robotic Arm for Reach and Grasp Tasks", *Sci. Rep.*, vol. 6, nº November, pp. 1–15, 2016.

[3] A. Barachant, "Code and documentation for the winning sollution to the Grasp-and-Lift EEG Detection challenge". [En línea]. Disponible en: https://github.com/alexandrebarachant/Grasp-and-lift-EEG-challenge. [Accedido: 22-abr-2018].

[4] M. D. Luciw, E. Jarocka, y B. B. Edin, "Multi-channel EEG recordings during 3,936 grasp and lift trials with varying weight and friction", *Sci. Data*, vol. 1, pp. 1–11, 2014.

[5] INEI, "Primera Encuesta Nacional Especializada sobre Discapacidad 2012", 2014.

[6] Anonymous, "My Husband's Story - The ALS Association", 2013. [En línea]. Disponible en: http://www.alsa.org/about-als/2013-aam/stories/2013-aam-stories-427038871.html. [Accedido: 29-mar-2018].

[7] J. L. Collinger, B. Wodlinger, y A. B. Schwartz, "7 Degree-of-Freedom Neuroprosthetic Control By an Individual With Tetraplegia", *Lancet*, vol. 381, nº 9866, pp. 557–564, 2012.

[8] M. Mulas, M. Folgheraiter, y G. Gini, "An EMG-Controlled Exoskeleton for Hand Rehabilitation", en *9th International Conference on Rehabilitation Robotics, 2005. ICORR 2005.*, 2005, pp. 371–374.

[9] J. Ugaz, "Decana del Colegio de Enfermeros: 'En el Perú hay 12 enfermeras por cada 10 mil habitantes'", *El Correo*, Lima, 14-feb-2018.

[10] Chau Delgado, Juan Manuel, "SELECCION DE TAREAS PREDEFINIDAS PARA UN ROBOTASISTENCIAL PARA PERSONAS DISCAPACITADAS A TRAVESDE UNA INTERFAZ CEREBRO-COMPUTADOR UTILIZANDOP300", 2018.

[11] Kinova Inc., "MICO^2 User guide". p. 39, 2017.

[12] A. Campeau-lecours, "Kinova Modular Robot Arms for Service Robotics Applications", nº

July 2017, 2018.

[13] Brainworks, “What are Brainwaves ? Types of Brain waves | EEG sensor and brain wave – UK”, 2007. [En línea]. Disponible en: http://www.brainworksneurotherapy.com/what-are-brainwaves. [Accedido: 29-mar-2018].

[14] G. Pfurtscheller, “EEG Rhythms - Event-Related Desynchronization and Synchronization”, en *Rythms in Physiological Systems*, Springer, Berlin, Heidelberg, 1991, pp. 289–296.

[15] I. Vigué, “Introduction to Mu Waves”, 2016. [En línea]. Disponible en: https://irenevigueguix.wordpress.com/2016/04/18/introduction-to-mu-waves-for-bci/. [Accedido: 29-mar-2018].

[16] OpenStax College, “Anatomy & Physiology”. [En línea]. Disponible en: https://commons.wikimedia.org/w/index.php?curid=30148008 [Accedido: 12-jul-2020].

[17] J. Brownlee, “Linear Discriminant Analysis for Machine Learning - Machine Learning Mastery”, 2016. [En línea]. Disponible en: https://machinelearningmastery.com/linear-discriminant-analysis-for-machine-learning/. [Accedido: 29-mar-2018].

[18] C. Stergiou y D. Siganos, “Neural Networks”.

[19] Stanford, “Unsupervised Feature Learning and Deep Learning Tutorial”. [En línea]. Disponible en: http://ufldl.stanford.edu/tutorial/supervised/ConvolutionalNeuralNetwork/. [Accedido: 29-mar-2018].

[20] C. Olah, “Understanding LSTM Networks -- colah’s blog”, 2015. [En línea]. Disponible en: http://colah.github.io/posts/2015-08-Understanding-LSTMs/. [Accedido: 29-mar-2018].

[21] Mlxtend, “Stacking CV Classifier”. [En línea]. Disponible en: http://rasbt.github.io/mlxtend/user_guide/classifier/StackingCVClassifier/ [Accedido: 12-jul-2020].

[22] Scikit-learn, “Receiver Operating Characteristic (ROC)”. [En línea]. Disponible en: https://scikit-learn.org/stable/auto_examples/model_selection/plot_roc.html#sphx-glr-auto-examples-model-selection-plot-roc-py. [Accedido: 12-jul-2020].

[23] J. Rickert y J. Fischer, “BCI apparatus for stroke rehabilitation”, US8509904B2, 2013.

[24] M. Linderman y V. Rupasov, “Method and apparatus for biometric analysis using EEG and

EMG signals", US9155487B2, 2005.

[25] D. Planelles, E. Hortal, Á. Costa, A. Úbeda, E. Iáñez, y J. M. Azorín, "Evaluating classifiers to detect arm movement intention from EEG signals", *Sensors (Switzerland)*, vol. 14, nº 10, pp. 18172–18186, 2014.

[26] A. Barachant, S. Bonnet, M. Congedo y C. Jutten, "Classification of covariance matrices using a Riemannian-based kernel forBCI applications", *Neurocomputing, Elsevier*, 112, pp. 172-178, 2013.

[27] A. Barachant, y M. Congedo, "A Plug&Play P300 BCI Using Information Geometry", *arXiv*, 2014.

[28] g. te. M. E. GmbH, "G.Nautilus". p. 116, 2016.

[29] Kinova Robotics, "GitHub - Kinovarobotics/kinova-ros: Official ROS packages for Kinova robotic arms". [En línea]. Disponible en: https://github.com/Kinovarobotics/kinova-ros. [Accedido: 23-may-2018].

[30] Gtec, "Gnautilus research". [En línea]. Disponible en: https://www.gtec.at/product/gnautilus-research/ [Accedido: 12-jul-2020].

[31] Ripley, "Lenovo laptop ideapad 520 15.6; core I7 1TB 12GB 4GB | Ripley". [En línea]. Disponible en: https://simple.ripley.com.pe/lenovo-laptop-ideapad-520-156-core-i7-1tb-12gb-4gb-2004195275364p. [Accedido: 23-may-2018].

[32] ROS, "Kinova MICO". [En línea]. Disponible en: https://robots.ros.org/mico/. [Accedido: 12-jul-2020].

[33] International Air Transport Association, "2017 Lithium Battery Guidance Document", pp. 1–20, 2017.

[34] Joel Lee, "How Much Power Does Your PC Need?", *19/05/2017*, 2017. [En línea]. Disponible en: https://www.makeuseof.com/tag/power-pc-technology-explained/. [Accedido: 25-may-2018].

[35] E. Maby, "Practical Guide to Performing an EEG Experiment", en *Brain-Computer Interfaces 2*, Hoboken, NJ, USA: John Wiley & Sons, Inc., 2016, pp. 163–177.

[36] PennState Lehigh Valley, "Write a Protocol". [En línea]. Disponible en:

http://www2.lv.psu.edu/jxm57/irp/prot.htm. [Accedido: 22-may-2018].

[37] Kaggle, “Grasp and lift eeg detection data”. [En línea]. Disponible en: https://www.kaggle.com/c/grasp-and-lift-eeg-detection/data [Accedido: 12-jul-2020].

[38] A. Campeau-lecours, “Kinova Modular Robot Arms for Service Robotics Applications”, nº July 2017, 2018.

[39] K. Varszegi, “Comparison of algorithms for detecting hand movement from EEG signals”, *2016 IEEE Int. Conf. Syst. Man, Cybern. SMC 2016 - Conf. Proc.*, pp. 2208–2213, 2017.

[40] Quanser, “Kinova 6 DOF MICO Read”. [En línea]. Disponible en: https://www.kaggle.com/c/grasp-and-lift-eeg-detection/data [Accedido: 12-jul-2020].

[41] Amazon, “Amazon.com: Kinova Mico Robotic Arm: Electronics”. [En línea]. Disponible en: https://www.amazon.com/Kinova-Mico-Robotic-Arm/dp/B00WT7TH1A. [Accedido: 23-may-2018].

[42] Ponte en carrera, “¿Cuánto ganan los profesionales universitarios?”. [En línea]. Disponible en: https://www.ponteencarrera.pe/pec-portal-web/inicio/como-va-el-empleo. [Accedido: 15-feb-2020].

ANEXOS

ANEXO A: PROGRAMA DEL ALGORITMO DE TRADUCCIÓN

A continuación, los programas del algoritmo de traducción para clasificar las señales EEG en uno de 6 eventos.

classify.py – parte 1

```
# Este programa recibe un archivo .csv con las señales EEG como entradas.
# Luego, predice el evento más probable y lo envía al brazo robot.
import numpy as np
import os, sys
if __name__ == '__main__' and __package__ is None:
    filePath =os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)
from classifyClf import classifyClf
from classifyCNN import classifyCNN
from classifyRNN import classifyRNN

try:
    dataFileTest = sys.argv[1]
    subj = sys.argv[2]
except:
    raise("Invalid arguments")
modelClfNames = ['CovAlex_500','CovAlex_500_1-15','CovAlex_500_20-
35','CovAlex_500_70-150', 'CovAlex_250_35Hz', 'CovAlex_500_35Hz',
'CovERP_dist', 'CovAlex_500_poly','CovAlex_500_1-
15_poly','CovAlex_500_20-35_poly', 'CovAlex_500_70-150_poly',
'CovAlex_250_35Hz_poly', 'CovAlex_500_35Hz_poly', 'CovERP_dist_poly',
'CovRafal_256_35Hz', 'CovRafal_500_35Hz', 'CovAlex_All',
'CovAlex_old_All','FBL','FBL_delay','FBLC_256pts','FBLCR_256']
modelCNNNames=['cnn_script_2D_30Hz','cnn_script_1D_30Hz','cnn_script_1D_5
Hz','cnn_script_1D_7-30Hz']
modelLvl2Names=['xgb_bags','xgb_bags_delay','xgb_bags_model']

# LVL 1
for yamlFile in modelClfNames:
    classifyClf(dataFileTest, yamlFile, subj)
for yamlFile in modelCNNNames:
    classifyCNN(dataFileTest, yamlFile, subj)
classifyRNN(dataFileTest, 'RNN_FB_delay4000', subj)
# LVL 2
classifyEns('xgb_short', subj)
for yamlFile in modelLvl2Names:
    classifyEnsBagsModel(yamlFile, subj)
# LVL 3
classifyFinal('Safe1',subj)
```

classify.py – parte 2

```
ordenEvento=[5000,5000,5000,5000,5000,5000]
eventos=["0","1","2","3","4","5"]
resultados=np.load("../lvl3/Safe1.npy")
for i in range(6):
    for j in range(len(resultados[0])):
         if(resultados[0][j][i]>=0.5):
               ordenEvento[i]=j
               break
for i in range(6):
    indexMin=ordenEvento.index(min(ordenEvento))
    if ordenEvento[indexMin]<5000:
        cliente(eventos[indexMin])
        ordenEvento[indexMin]=5000
```

classifyClf.py – parte 1

```
from sklearn.externals import joblib
import numpy as np
import os, sys, yaml, time
if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)
from preprocessing.aux import load_raw_data, makeDir
import warnings
warnings.filterwarnings('ignore')

# Esta función predice los eventos con modelos de machine learning
def classifyClf(dataFileTest, yamlFile, subj):
    start_time = time.time()
    # Se cargan las arquitecturas de los modelos. En este caso, son más
    # de una
    yml = yaml.load(open('../lvl1/models/'+yamlFile+'.yml'))
    for pkg, functions in yml['imports'].iteritems():
       stri = 'from ' + pkg + ' import ' + ','.join(functions)
       exec(stri)
    fileName = yml['Meta']['file']
    cache_preprocessed = yml['Meta']['cachePreprocessed']
    clfNumber = 0
    for mdl in yml['Models']:
       clfNumber+=1
    if 'Preprocessing' in yml.keys():
        existPre = True
```

classifyClf.py – parte 2

```
        for key in yml['Preprocessing']:
            for e in key:
                if e=='NoneTransformer':
                    existPre = False
    else:
        existPre = False
    if 'PostPreprocessing' in yml.keys():
        existPost = True
    else:
        existPost = False

    print("Loading data "+fileName)
    dataTest = load_raw_data(subj, dataFileTest)
    print("Processing...")
    testPreprocessed = None

    # Preprocesamiento
    if existPre == True:
        preprocessing =
            joblib.load('../lvl1/procesamiento/%s/sub%s/sub%s_preproces
      sing.plk' %(fileName, subj, subj))

        if('CovAlex' in yamlFile):
          covmean=
            np.load('../lvl1/covmeans/%s/sub%s/sub%s_covmeans.npz' %
            (fileName, subj, subj))
           preprocessing.named_steps["distancecalculatoralex"].
                  mdm.covmeans = covmean['arr_0']
        elif('CovRafal' in yamlFile):
           covmean=
            np.load('../lvl1/covmeans/%s/sub%s/sub%s_covmeans.npz' %
            (fileName, subj, subj))
           preprocessing.named_steps["distancecalculatorrafal"].
                  mdm.covmeans = covmean['arr_0']
        elif('CovERP' in yamlFile):
           covmean=
np.load('../lvl1/covmeans/%s/sub%s/sub%s_covERP.npz' % (fileName, subj,
subj))
           preprocessing.named_steps["erpdistance"].MDM.covmeans =
covmean['arr_0']
        testPreprocessed = preprocessing.transform(dataTest)
        fileSaved = '../lvl1/feat_file'
        makeDir(fileSaved)
```

classifyClf.py – parte 3

```
    # Cargando preprocesamiento previo
    if 'addPreprocessed' in yml['Meta']:
        addPreprocessed = yml['Meta']['addPreprocessed']
        print("Loading preprocessed...")
        for feat_name in addPreprocessed:
            featFile =
'../lvl1/feat_file/%s_preprocess.npy' % (feat_name)
            if os.path.isfile(featFile):
                feat = np.load(featFile)
                if testPreprocessed is None:
                    testPreprocessed = feat
                else:
                    testPreprocessed = np.c_[testPreprocessed
, feat]
                feat = None
            else:
                raise ValueError("File %s does not exist" % featFile)

    # Postprocesamiento
    if existPost == True:
        postprocessing =
                  joblib.load('../lvl1/procesamiento/%s/sub%s/
                  sub%s_postprocessing.plk' %(fileName, subj, subj))
        testPreprocessed = postprocessing.transform(testPreprocessed)
    # Predicción por clasificador y por columna
    tot_predictions = []
    for i in range(clfNumber):  # número de clasificadores
        predictions = []
        for col in range(6):    # 6 columnas (6 eventos)
            clf =
                  joblib.load('../lvl1/modelo/%s/sub%s/clf%d/sub%s_clf%
            d_col%d.plk' %(fileName, subj, i, subj, i, col))
            print("Predicting column number %d" %(col))
            predictions.append(clf.predict_proba(testPreprocessed)
                               [:, 1])
        tot_predictions.append(predictions)

    fileSaved = '../lvl1/predicciones/%s/' %(fileName)
    makeDir(fileSaved)
    np.save(fileSaved + 'preds_%s.npy' %(fileName), tot_predictions)
    print("Time: %s seconds" %(time.time()-start_time))
```

classifyCNN.py – parte 1

```
from sklearn.externals import joblib
import numpy as np
import os, sys, yaml, time
if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)
from preprocessing.cnnAux import create_net, IndexBatchIterator,
TestSource, Source, preprocessData
# Esta función recibe como parámetros la señal EEG, el nombre del
# modelo  y el número de sujeto
def classifyCNN(dataFileTest, yamlFile, subj):
    start_time = time.time()
    # Cargando arquitectura del modelo
    yml = yaml.load(open('../lvl1/models/'+yamlFile+'.yml'))
    fileName = yml['Meta']['file']
    print("Loading model " + fileName)
    filt2Dsize = yml['filt2Dsize'] if 'filt2Dsize' in yml.keys() else 0
    filters = yml['filters']
    delay = yml['delay']
    skip = yml['skip']
    if 'bags' in yml.keys():
        bags = yml['bags']
    else:
        bags = 3
    # Preprocesamiento por filtro de bancos y predicciones
    preds_tot = []
    max_epochs = 100
    np.random.seed(67534)
    for bag in range(bags):
        print("Predicting bag %d..." %(bag))
        test_source = TestSource(subj, dataFileTest, filters, fileName)
        indices = np.arange(len(test_source.data))
        net = create_net(test_source, filt2Dsize,
            max_epochs=max_epochs, train_val_split=False)
        net.load_params_from(np.load('../lvl1/modelo/%s/sub%s/sub%s_CNN
      _bags%d.npz' % (fileName, subj, subj, bag)))
        preds = net.predict_proba(indices)
        preds_tot.append(preds)
    preds_total = np.mean(preds_tot, axis=0)
    fileSaved = '../lvl1/predicciones/%s/' %(fileName)
    makeDir(fileSaved)
    np.save(fileSaved + 'sub%s_CNN_predictions.npy' %(subj),
preds_total)
    print("Time: %s seconds" %(time.time()-start_time))
```

classifyRNN.py – parte 1

```
from sklearn.externals import joblib
import numpy as np
import os, sys, yaml, time
from ensembling.NeuralNet import NeuralNet
from preprocessing.aux import load_raw_data, makeDir
if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)

# Esta función predice el evento con una red neuronal recurrente
def classifyRNN(dataFileTest, yamlFile, subj):
    start_time = time.time()
    # Cargando arquitectura del modelo
    yml = yaml.load(open('../lvl1/models/'+yamlFile+'.yml'))
    for pkg, functions in yml['imports'].iteritems():
       stri = 'from ' + pkg + ' import ' + ','.join(functions)
       exec(stri)
    fileName = yml['Meta']['file']
    training_params = yml['Training']
    architecture = yml['Architecture']
    delay = training_params['delay']
    skip = training_params['skip']
    parts_train = training_params['parts_train']
    parts_test = training_params['parts_test']
    smallEpochs = training_params['smallEpochs']
    majorEpochs = training_params['majorEpochs']
    checkEveryEpochs = training_params['checkEveryEpochs']
    subsample = training_params['subsample']
    # Cargando data EEG y modelos
    print("Loading data "+fileName)
    dataTest = load_raw_data(subj, dataFileTest)
    preprocessing =
joblib.load('../lvl1/procesamiento/%s/sub%s/sub%s_preprocessing.plk'
                                        %(fileName, subj, subj))
    postprocessing =
joblib.load('../lvl1/procesamiento/%s/sub%s/sub%s_postprocessing.plk'
                %(fileName, subj, subj))
    model = NeuralNet(None, architecture, training_params,
                      partsTrain=parts_train,partsTest=parts_test,
                      delay=delay,skip=skip,subsample=subsample,
                      majorEpochs=majorEpochs,smallEpochs=smallEpochs,
                      checkEveryEpochs=checkEveryEpochs)
    model.loadModel(fileName,subj,1)
```

classifyRNN.py – parte 2

```
    # Procesamiento y prediccion
    print("Processing...")
    testPreprocessed = preprocessing.transform(dataTest)
    print("Predicting...")
    predictions = model.predict_proba(testPreprocessed)
    fileSaved = '../lvl1/predicciones/%s/sub%s' % (fileName, subj)
    makeDir(fileSaved)
    np.save(fileSaved + '/sub%s_RNN_predictions.npy'
                            %(subj), predictions)
    print("Time: %s seconds" %(time.time()-start_time))
```

classifyEns.py – parte 1

```
import numpy as np
import os
import sys
from ensembling.NeuralNet import NeuralNet
from keras.models import model_from_yaml

if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)

def _from_yaml_to_func(method, params):
    prm = dict()
    if params is not None:
        for key, val in params.iteritems():
            prm[key] = eval(str(val))
    return eval(method)(**prm)

# Esta función recibe como parámetros el nombre del modelo y el número
# de sujeto
def classifyEns(yamlFile, subj):
    start_time = time.time()
    # Se cargan datos de la arquitectura
    yml = yaml.load(yamlFile)
    fileName = yml['Meta']['file']
    print("Loading data " + fileName)
    modelName, modelParams = yml['Model'].iteritems().next()
    model_base = _from_yaml_to_func(modelName, modelParams)
    ensemble = yml['Model'][modelName]['ensemble']
```

classifyEns.py – parte 2

```
    # Se cargan las predicciones del nivel 1
    files = getLvl1ModelList()
    preds_lvl1 = OrderedDict()
    for f in files:
        loadPredictions(preds_lvl1, f[0], f[1], 1)

    for m in ensemble:
        assert(m in preds_lvl1)
    # Se crea el ensemble
    aggr = createEnsFunc(ensemble)
    dataTest = aggr(preds_lvl1)
    preds_test = None

    # Cargando modelo y prediciendo
    model = deepcopy(model_base)
    model.loadModel(fileName, 0, 2, 0)
    print("Predicting...")
    preds = model.predict_proba(dataTest)

    np.save('../lvl2/predicciones/%s/%s_Ens_predictions.npy'
                            %(fileName, fileName), [preds])
    print("Time: %s seconds" %(time.time()-start_time))
```

classifyEnsBags.py – parte 1

```
import numpy as np
import os
import sys
from ensembling.NeuralNet import NeuralNet
from keras.models import model_from_yaml
if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)
def _from_yaml_to_func(method, params):
    prm = dict()
    if params is not None:
        for key, val in params.iteritems():
            prm[key] = eval(str(val))
    return eval(method)(**prm)
```

classifyEnsBags.py – parte 2

```
# Esta función recibe como el nombre del modelo y el número de sujeto
def classifyEnsBags(yamlFile, subj):
    start_time = time.time()
    # Se cargan datos de la arquitectura
    yml = yaml.load(yamlFile)
    fileName = yml['Meta']['file']
    print("Loading data " + fileName)
    modelName, modelParams = yml['Model'].iteritems().next()
    model_base = _from_yaml_to_func(modelName, modelParams)
    ensemble = yml['Model'][modelName]['ensemble']
    bags = yml['Meta']['nbags']
    bagsize = yml['Meta']['bagsize']
    # Se cargan las predicciones del nivel 1
    files = getLvl1ModelList()
    preds_test = OrderedDict()
    for f in files:
        loadPredictions(preds_test, f[0], f[1], subj, 1)
    for m in ensemble:
        assert(m in preds_test)
    # Se crea el ensemble
    aggr = createEnsFunc(ensemble)
    dataTest = aggr(preds_test)
    preds_test = None

    # Cargando modelo y prediciendo
    for k in range(bags):
        model = deepcopy(model_base)
        model.loadModel(fileName, 0, 2, k)
        print("Predicting bag %d" + %k)
        preds += model.predict_proba(dataTest)/ bags

    np.save('../lvl2/predicciones/%s/%s_Ens_Bags%d_predictions.npy'
                %(fileName, fileName, bags), [preds])
    print("Time: %s seconds" %(time.time()-start_time))
```

classifyFinal.py – parte 1

```
import os
import sys
if __name__ == '__main__' and __package__ is None:
    filePath =
os.path.dirname(os.path.dirname(os.path.abspath(__file__)))
    sys.path.append(filePath)

import pandas as pd
import numpy as np
import yaml

from collections import OrderedDict
from sklearn.metrics import roc_auc_score
from sklearn.cross_validation import LeaveOneLabelOut
from preprocessing.aux import getEventNames
from utils.ensembles import createEnsFunc, loadPredictions
from ensembling.WeightedMean import WeightedMeanClassifier

def classifyFinal(yamlFile, subj)
    start_time = time.time()
    # Cargando arquitecturas de los modelos
    yml = yaml.load(yamlFile)
    fileName = yml['fileName']

    print("Loading data " + fileName)
    ensemble = yml['ensemble']
    models = []
    for m in mean_type:
        models.append(WeightedMeanClassifier(ensemble, mean=m,
verbose=verbose))

    # Se cargan las predicciones del nivel 2
    files = ensemble
    preds_val = OrderedDict()
    for f in files:
        loadPredictions(preds_val, f, [f], lvl=2)
    # Se crea el ensemble
    aggr = createEnsFunc(ensemble)
    dataTest = aggr(preds_val)
    preds_val = None

    print("Predicting...")
    preds = 0
```

classifyFinal.py – parte 2

```
    for m in range(len(models)):
        models[m].loadParameters(fileName)
        preds += models[m].predict_proba(dataTest) / len(models)
    print("Saving results...")
    sub = pd.DataFrame(data=p,index=ids,columns=cols)
    sub.to_csv('../lvl3/predicciones.npy', preds)
    print("Time: %s seconds" %(time.time()-start_time))
```

cliente.py

```
import socket
import sys

def cliente(data):
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 10000)
    print('Connecting to {} port {}'.format(*server_address))
    sock.connect(server_address)

    try:
        message = data
        print('Sending {!r}'.format(message))
        sock.sendall(message)
    finally:
        print('Closing socket')
        sock.close()
```

ANEXO B: PROGRAMA PARA COMANDAR EL BRAZO KINOVA

A continuación, los programas para el comando del brazo Kinova.

Programa servidor-kinova

```
import socket, sys, rospy
from std_msgs.msg import Int8

def talker(data):
    msg = Int8(int(data))
    rospy.loginfo(msg)
    pub.publish(msg)
if __name__ == '__main__':
    pub = rospy.Publisher('eventoKinova', Int8, queue_size=1)
    rospy.init_node('clienteKinova')#, anonymous=True)
    sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_address = ('localhost', 10000)
    sock.bind(server_address)
    sock.listen(1)
    while True:
        connection, client_address = sock.accept()
        try:
            print('Connection from', client_address)
            while True:
                data = connection.recv(8)
                if data:
                    print('Received {!r}'.format(data))
                    try:
                        talker(data)
                    except rospy.ROSInterruptException:
                        pass
                else:
                    break
        finally:
            connection.close()
```

Programa comandarBrazo – parte 1

```
#include <ros/ros.h>
#include <math.h>
#include <std_msgs/Int8.h>
#include <sys/socket.h>
#include <boost/bind/bind.hpp>
#include <actionlib/client/simple_action_client.h>
#include <kinova_msgs/ArmPoseAction.h>
#include <kinova_msgs/SetFingersPositionAction.h>

typedef actionlib::SimpleActionClient<kinova_msgs::ArmPoseAction>
KinovaClient;
typedef actionlib::SimpleActionClient<kinova_msgs::SetFingersPositionAction>
GripperClient;
const float posiciones[6][3] = {{0.25,-0.24,0.09},{0.4,-0.24,0.09},{0.4,-
0.24,0.09},{0.4,-0.24,0.35},{0.4,-0.24,0.09},{0.4,-0.24,0.09}};
const int estadoGripper[6][2] =
{{0,0},{0,0},{5100,5100},{5100,5100},{5100,5100},{0,0}};

void interpetarComando(const std_msgs::Int8ConstPtr &msg){
   ROS_INFO_STREAM("Connecting to clients..");
   KinovaClient client1("/m1n6s200_driver/pose_action/tool_pose", true);
   GripperClient client2("/m1n6s200_driver/fingers_action/finger_positions",
true);
   client1.waitForServer(ros::Duration(1.0));
   client2.waitForServer(ros::Duration(1.0));
   kinova_msgs::ArmPoseGoal goal;
   kinova_msgs::SetFingersPositionGoal fingersGoal;

   int evento;
   evento = msg->data;
   goal.pose.header.frame_id = "m1n6s200_link_base";
   goal.pose.pose.position.x = posiciones[evento][0];
   goal.pose.pose.position.y = posiciones[evento][1];
   goal.pose.pose.position.z = posiciones[evento][2];
   goal.pose.pose.orientation.x = 0.4998;
   goal.pose.pose.orientation.y = 0.4999;
```

Programa comandarBrazo – parte 2

```
    goal.pose.pose.orientation.z = 0.4998;
    goal.pose.pose.orientation.w = 0.5005;
    client1.sendGoal(goal);
    client1.waitForResult();
    if(client1.getState()==actionlib::SimpleClientGoalState::SUCCEEDED){
        ROS_INFO_STREAM("Message sended");
    }
    else if(client1.getState()==actionlib::SimpleClientGoalState::ABORTED){
        ROS_INFO_STREAM("Aborted");
    }
    else{
        ROS_INFO_STREAM("Message failed to send");
    }
    fingersGoal.fingers.finger1 = estadoGripper[evento][0];
    fingersGoal.fingers.finger2 = estadoGripper[evento][1];
    fingersGoal.fingers.finger3 = 0; // El tercer dedo debe ser cero
    client2.sendGoal(fingersGoal);
    client2.waitForResult();
    if(client2.getState()==actionlib::SimpleClientGoalState::SUCCEEDED){
        ROS_INFO_STREAM("Message sended");
    }
    else if(client2.getState()==actionlib::SimpleClientGoalState::SUCCEEDED){
        ROS_INFO_STREAM("Aborted");
    }
    else{ ROS_INFO_STREAM("Message failed to send"); }
}
int main(int argc, char **argv){
    ros::init(argc, argv, "comandarBrazo");
    ros::NodeHandle nh;
    ROS_INFO_STREAM("Node initialized.");
    // Subscriber
    ros::Subscriber sub = nh.subscribe("eventoKinova", 1,
        &interpetarComando);
    ros::spin();
    return 0;
}
```