Licenciatura en Música y Tecnología

PulsAr: Desarrollo de un sistema mecatrónico de percusión, construido con materiales recuperados

Tesis de Grado de Nicolás Miñán

Director de tesis:
Martín Matus Lerner

Fecha de presentación:
9 de noviembre de 2020

Agradecimientos

Este escrito es la síntesis de un trabajo de casi 2 años, con el cual se cierra un camino que iniciara en 2006, para abrir otros. Este esfuerzo personal no hubiera sido posible sin la contribución y la ayuda de muchxs a quienes quiero afectuosamente agradecer.

A mis compañerxs-amigxs de militancia de todos estos años, con quienes peleamos codo acodo por un mundo verdaderamente justo. En especial a Pablito Marzano, a Aldo y Oscar Mananicci, quienes aportaron instrumentos, ideas y apoyo para este trabajo; y al querido Tata, por estar siempre, incondicionalmente.

A Pablo Pereyra, quien desde la escuela amplió mi horizonte en la música y la tecnología y cuya influencia resuena hasta hoy en día. Y a mis compañeros de banda, Pablo y Diego con quienes dimos los primeros pasos en la aventura del sonido.

A Lucio y a Leandro, los mejores colegas, compañeros y amigos que me regaló está carrera, fuente de empuje e inspiración permanente.

A los enormes maestros con los que pude aprender en esta casa. En particular, a Orlando Musumeci, Pablo Cetta y Esteban Calcagno, cuyas enseñanzas están plasmadas en este trabajo, y en especial a Martín Matus, mi director de tesis, de beca, y maestro cotidiano en el bello camino de la investigación. Y por supuesto a la Universidad Nacional de Quilmes, por confiar en este trabajo y abrirme tantas puertas.

A Romina y Charo quienes me ayudaron y ayudan a comprenderme un poco más.

A mi tío Daniel (a quien le robé mí primer cable MIDI hace más de 20 años), por los discos y por enseñarme a jugar a la música.

Y muy especialmente a mi familia. A mis viejxs, quienes siempre estuvieron y están ahí, por enseñarme a observar y a trabajar rigurosamente, a ser empático y a nunca conformarme. A mis hermanxs, de quienes aprendo cada día, por su inmenso cariño. A lxs cuatro, gracias por su apoyo y empuje permanente para terminar esta carrera.

Dedico este trabajo a mis abuelas, María y Lilí, por su amor infinito. Y a todxs aquellxs que luchan cada día por una educación pública, gratuita y de calidad.

8 de octubre de 2020

Índice

Resumen

El presente trabajo consiste en el d*esarrollo de un sistema mecatrónico de percusión, construido con materiales recuperados*. Esto es la utilización de dispositivos electrónicos, electromecánicos y diversos materiales reciclados, en mayor medida extraídos de "chatarra tecnológica" doméstica (impresoras, fotocopiadoras, máquinas de fax, CPUs, etc.), para la construcción de un artefacto capaz de excitar, mediante movimientos controlables, objetos en función de la generación de sonido. El "cerebro" del sistema está basado en microcontroladores Esspresif ESP32 (símil Arduino) y utiliza protocolo MIDI como mecanismo de control digital.

En primer lugar, el proyecto busca aportar al campo de desarrollo de instrumentos musicales robóticos, particularmente al segmento de artefactos de percusión. Dentro de este marco, resultan de principal interés los resultados *puramente acústicos*, es decir no mediados por parlantes, junto con un abordaje sonoro no convencional.

En segundo lugar, busca contribuir al desarrollo de técnicas, criterios, experiencias del llamado "hardware hacking" orientado a música y arte sonoro, haciendo foco en la recuperación, reciclado y reutilización de la gran cantidad de chatarra tecnológica generada en las zonas urbanas a partir del descarte de artefactos domésticos. En sentido, se pone en relieve la implementación de estrategias de construcción de bajo presupuesto y tecnologías de software y hardware libre.

El proyecto toma como nombre PulsAr, acrónimo de "Pulso Argentino", buscando unir conceptualmente la naturaleza del material sonoro generado y su utilización como herramienta que opera en el *campo de la rítmica y la percusión*; con el hecho de ser un desarrollo *con los pies en la tierra*, en un territorio determinado: la Argentina de principios de s.XXI, con sus particulares condiciones socio-culturales.

Cabe mencionar que gran parte de este trabajo ha sido realizado en el marco de la Beca de formación inicial en la investigación, convocatoria 2018, otorgada por la Secretaría de Investigación de la UNQ. A su vez, que el mismo ha sido adaptado para la participación en el II Premio a la Innovación en Artes y Tecnologías de la Escuela Universitaria de Artes (UNQ) 2019, resultando ganador en la categoría correspondiente con la propuesta la propuesta "Puls.Ar: ¡Hacé vos mismx tu robot musical!".

Palabras clave: robótica, mecatrónica, percusión, arduino, reciclado, hardware-hacking

Introducción

1. Idea general y motivaciones del proyecto

El dispositivo consiste en un conjunto o arreglo (*array*) de objetos físicos de distinto tamaño y material, los cuales son excitados de múltiples formas por motores electromecánicos, mediante diversos tipos de complementos o apliques. Dichos actuadores son impulsados por drivers alimentados por fuentes de alimentación externa (5, 12 y 24 voltios) y manejados por microcontroladores[1] de enfoque Open Hardware (Arduino, Esspressif ESP32). Estos últimos son el cerebro del dispositivo, los cuales reciben mensajes de control usando protocolo MIDI provenientes de un secuenciador o controlador para accionar los motores y generar sonido.

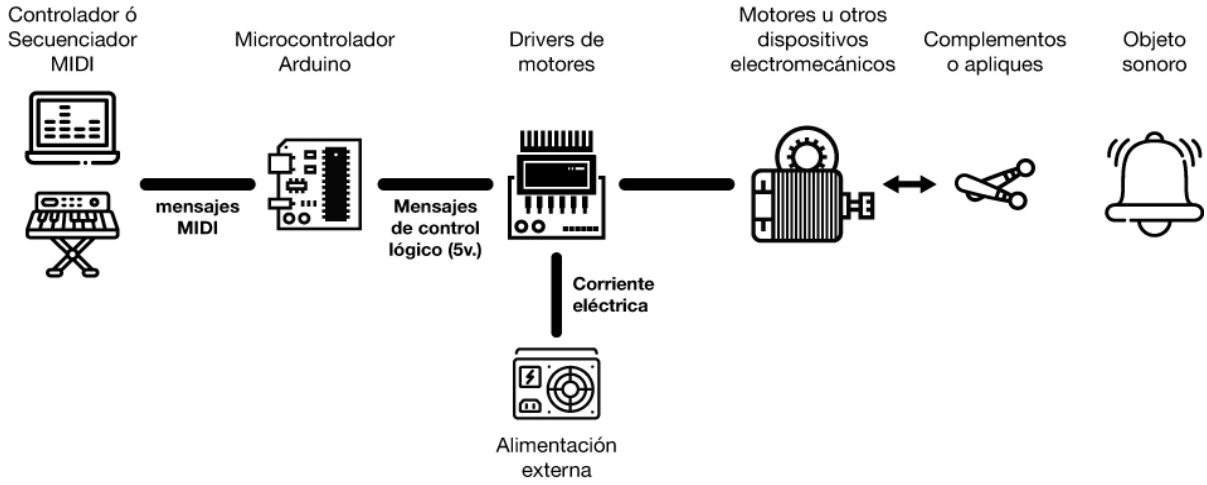

Fig. 1: Cadena de dispositivos del sistema a construir

Los artefactos obtenidos califican como variantes de dispositivos mecatrónicos, autómatas o robots. A lo largo del trabajo nos referimos a estos como IMR: *instrumentos musicales robóticos*. Cabe señalar que se trata de robots no-antropomórficos, es decir dispositivos que no emulan la forma humana.

Tanto en lo que hace a los dispositivos electrónicos y electromecánicos, a los elementos estructurales de construcción de soportes, gabinetes, como a los objetos sonoros cuyo cuerpo se excita mecánicamente para producir sonido, se utilizó *la mayor cantidad posible de material obtenido de artefactos tecnológicos, u objetos de descarte*: chatarra, basura, etc. En combinación con un porcentaje de insumos que indefectiblemente debió adquirirse para este desarrollo, el material recuperado es el componente vital, la marca identitaria, estructural-funcional y estética, del sistema construido.

1 En adelante, MC

En primer lugar, motivó a llevar adelante este proyecto el realizar una contribución a la exploración y desarrollo de recursos tecnológico-sonoro-musicales desde un enfoque relativamente no convencional. En una época donde las herramientas de audio digital dominan la escena y están a la vanguardia en la exploración y producción músico-tecnológica, resulta de interés sondear un camino alternativo, que se sirva de la creciente difusión de herramientas de software y hardware abierto, para encarar el diseño sonoro y la composición usando recursos digitales electrónicos y electromecánicos, pero apuntando específicamente a resultados acústicos o electro-acústicos[2].

En este punto, se centró la atención en darle lugar, en la experiencia musical con dispositivos digitales-electrónicos-mecánicos, al fenómeno de escucha de una fuente sonora original en vivo, con ninguna o la menor cantidad de mediaciones posibles entre esta y el aparato auditivo. Es decir, evitando todo cuanto se pueda el uso de parlantes u otros dispositivos de reproducción del material sonoro. Experiencia que se ve enriquecida por el acompañamiento visual de la fuente sonora siendo excitada. Osea, en el fenómeno de escucha presencial, *in-situ*, con el material sonoro generándose irrepetiblemente allí, potenciando la relación humana entre desarrolladores, colegas, oyentes, espacio, tecnología y sonido. En definitiva, y lejos de cualquier concepción conservadora, romántica (o metafísica), se busca trabajar, en la relación humana artística, con el "aura" de la que hablaba W. Benjamin (Benjamin, 1936).

En segundo lugar, se buscó aprovechar positivamente la creciente cuota de estocástica, el margen de no-control que brindan los medios físicos, corpóreos, tangibles, en la construcción de instrumentos y dispositivos musicales, como fuente de riqueza tímbrica. No-control o comportamiento no-lineal que se potencia usando materiales de descarte, con una *historia a cuesta*.

A su vez, la realización de este proyecto se situó concretamente en tiempo y espacio: en la difícil realidad socio-económica que atraviesa la Argentina desde hace años, potenciada por la pandemia de Covid19 que abarcó el año 2020. Es así que, en tercer lugar, el trabajo buscó contribuir a la sistematización, documentación y difusión de diversas estrategias que incorporen bajo presupuesto en los procesos de desarrollo y tecnologías de hardware libre como Arduino, para la realización de dispositivos tecnológicos, en particular en lo que hace a mecatrónica, electrónica aplicada a la música y afines. Esto en el marco de contribuir al desarrollo, acceso y socialización de conocimiento de dominio público.

Ligado a lo anterior, en cuarto lugar, el trabajo apuntó a explorar formas de reconvertir a materia prima funcional, "vital", los residuos generados por la matriz de consumo tecnológico doméstico, marcada por la obsolescencia programada. Esto, a fin de visibilizar un esquema de relaciones sociales ineficiente desde un punto de vista económico-social, e insustentable y perjudicial desde un enfoque socio-ambiental. Así, el proyecto buscó sumar ejemplos de que, lo que es descarte tecnológico según los patrones de consumo promedios, es en realidad material útil, fácilmente recuperable o reciclable.

En síntesis, el proyecto tuvo como objetivos:

- Alcanzar una metodología eficaz para la recuperación y reutilización de componentes y materiales electrónicos a partir de "chatarra tecnológica".

2 En línea con las categorías "Pure acoustics" (Puramente acústicos) y "Electro-Acoustics" (Electro-acústicos) introducida por Godfried-Willem Raes para organizar su vasta producción de IMRs (Raes, 1993).

- Establecer estrategias fiables para la generación de sonido usando dispositivos mecatrónicos, con riqueza y heterogeneidad tímbrica, las cuales pudieran ser reutilizadas en proyectos futuros más ambiciosos.
- Desarrollar diseños de circuitos para drivers de motores y otros, de buen rendimiento y bajo presupuesto, y que estos puedan estar disponibles, de manera documentada y con un perfil didáctico, para otros proyectos.
- Hacer de dominio público tanto los códigos fuente del software creado como los diagramas esquemáticos del hardware confeccionado.
- Aportar a la comunidad de la EUDA – UNQ un conjunto de instrumentos musicales que pueda ser integrado a composiciones y/o utilizado en performances en vivo.
- Elaborar algunos ejemplos musicales que permitan mostrar las capacidades expresivas del sistema.

2. Sobre el proceso de trabajo

El trabajo se dividió cronológicamente en cinco momentos fundamentales, bien distinguibles: (1) Investigación preliminar y planificación, (2) Ignición del proyecto, (3) Prototipado, (4) Construcción definitiva, y (5) Sistematización de conclusiones. Se puntualizan a continuación los aspectos centrales de cada etapa de trabajo.

2.1. Investigación y planificación

Incluyó la elaboración del plan tesis y de beca, la identificación de los principales momentos del desarrollo, las tareas que implicaría cada uno. También, la construcción del marco teórico inicial a partir de una lectura introductoria de autores referentes en la temática (Kapur, Raes). Finalmente, una primera introducción práctica en una parte fundamental del desarrollo, referida a la recolección y desarme de una primer lote de artefactos electrónicos.

2.2. Ignición del proyecto

Centrada en la consecución “a gran escala” de chatarra tecnológica y materiales que servirían para el desarrollo del proyecto. Posteriormente, su clasificación y recuperación de material útil: principalmente motores, pero también cables, componentes electrónicos de alto valor y material estructural (metal, plástico, tornillos, tuercas, etc.). Así también, la recuperación de motores, drivers, fuentes de alimentación, su restauración y acondicionamiento, y la consecución de especificaciones técnicas del fabricante. Esto, junto a una primera fase de recolección y reciclado de objetos/fuentes sonoras.

2.3. Prototipado

Consistió en la creación de primeras versiones de la programación, la electrónica y los dispositivos mecánicos, todo a fin de conseguir una primera resolución global a los diversos problemas del proyecto, obteniendo como resultado los primeros percutores / actuadores, a partir de motores eléctricos y diversos complementos o apliques para la excitación de cuerpos sonoros y finalmente, los primeros instrumentos robóticos construidos con material recuperado. Se probaron múltiples estrategias de producción sonora, percutores, materiales de construcción,

etc. resultando en un proceso de selección y descarte que marcaría el camino de la etapa de elaboración definitiva.

A su vez, desarrollo de una primera versión del sistema de control digital basado en MIDI, el cual permitiría su integración desde los comienzos del desarrollo con entornos de producción musical.

El proceso de prototipado ocupó aproximadamente la mitad del tiempo global invertido en el desarrollo del proyecto. Es importante destacar que esta etapa fue subdimensionada al momento de la planificación, al punto que terminó ocupando más del doble del tiempo del previsto. Esto, junto con los imponderables surgidos por la pandemia de Covid19 durante 2020, implicó una modificación importante del plan de trabajo inicialmente establecido.

2.4. Construcción definitiva

Centrada en la producción ulterior de la programación, dispositivos electrónicos y mecánicos y diversos elementos estructurales y de soporte del desarrollo (gabinetes, sujetadores, conexionado, etc), a partir de las conclusiones alcanzadas en la etapa de prototipado. Marcada sin dudas por una evolución del trabajo realizado en cada una de las esferas del trabajo: Construcción de los instrumentos tomando como base las nociones de sobre durabilidad, robustez, etc. (Maes, Raes y Rogers, 2011), pasaje a un enfoque de programación tipo POO[3], cambio en la placa microcontroladora, la IDE y el sistema de control de versiones y la construcción de un gabinete para alojar la electrónica del sistema. Incluyó también la adquisición de herramientas: amoladora, soldadora inverter, soldador de estaño, etc.

2.5. Sobre la etapa de elaboración de conclusiones

Sistematización del trabajo realizado: hipótesis, problemas, soluciones, conclusiones parciales y finales. Construcción del presente documento de tesis y escritos complementarios.

3 Programación orientada a objetos

Antecedentes y estado del arte

1. Antiguos instrumentos musicales autómatas

El actual desarrollo de IMRs encuentra sus antecedentes remotos en los antiguos instrumentos musicales autómatas de naturaleza mecánica. Se trata de las primeras generaciones de dispositivos cuya acción, es decir la generación sonoro-musical, no es producto de la mera prolongación directa del movimiento de extremidades humanas sino actividad mecánica independiente.

La construcción de este tipo de artefactos se remonta al menos hasta la edad media y el renacimiento, con instrumentos mecánicos construidos completamente a mano, donde la información musical se almacenaba en rodillos con púas (McVay, Kapur, 2012). Algunos de los más antiguos consistían en mecanismos de percusión automática implementados en carrillones de campanarios de iglesia en torno al siglo XIII (Long, Murphy, Kapur, 2015).

Ya en el siglo XVIII, la revolución industrial traería consigo la producción en masa de pianolas y artefactos similares. Dichos instrumentos, junto a organillos, carrillones y un puñado de artefactos más, serían la única forma de reproducir música no ejecutada directamente por humanos (McVay, Kapur, 2012). Finalmente en el siglo XIX, la utilización de principios neumáticos toma la delantera. Todos los instrumentos automáticos fabricados en ese momento (los antiguos órganos de Limonaire o Mortier, Pianolas, etc.) utilizaban rollos de papel o libros de cartón para la programación y se basaban en un funcionamiento neumático (Raes, 1987-2020). En su conjunto, estos constituyen los antepasados lejanos de los modernos instrumentos musicales mecatrónicos.

2. Autómatas modernos: Instrumentos musicales robóticos

Con el desarrollo de los dispositivos electromecánicos y electro-neumáticos y, tiempo después, con la aparición del MC, se potencia la práctica musical autómata y se expanden notablemente las posibilidades de desarrollo entre los modernos constructores de instrumentos musicales (Raes, 1987-2020). La capacidad expresiva (en cuanto a control parametrizable de los diversos aspectos del material sonoro generado: existencia de evento, duración, altura, intensidad, timbre, localización espacial) iría evolucionando conforme fuera avanzando la técnica y la destreza de constructores.

Hasta mitad del siglo XX, los artefactos serían puramente mecánicos o neumáticos (Kapur 2005). Poseerían un funcionamiento discreto y su expresividad se limitaría a encendido y apagado de notas en temporizaciones bastante precisas. El advenimiento posterior de la electromecánica y del MC extendería la versatilidad de los instrumentos. En palabras de G. W. Raes, "*los modernos IMRs controlados por computadora lograrían un control de los parámetros sonoro-musicales más fino que el que los seres humanos pudieran siquiera aspirar a conseguir*" (Maes, Raes, Rogers, 2011).

Con el incremento del poder computacional de los ordenadores entre 1990 y los 2000, los creadores de IMRs comenzaron a incorporar técnicas de inteligencia artificial en sus creaciones. (McVay, Kapur, 2012). La creciente masificación y accesibilidad del software y hardware libre junto con las inéditas posibilidades de difusión e intercambio de conocimiento que brinda Internet dieron lugar a la creación de gran cantidad de IMRs, de las más diversas formas y complejidades, alrededor de todo el mundo.

3. Principales trabajos en IMRs

La producción de IMRs en diversas partes del mundo es muy variada, con diversos enfoques y resultados. Entre las familias de instrumentos se encuentran: pianos robóticos, instrumentos basados en fonolas, instrumentos percusivos de incontables formas, instrumentos de cuerda, de viento, y dispositivos que combinan estrategias de generación de sonido puramente acústicas, mixtas, etc. (Kapur, 2005). Algunos de los proyectos que se encuentran a la vanguardia en la enorme familia global de producción de IMRs son:

- La Logos Robot Orchestra[4], junto con toda la inmensa obra en cuanto a construcción de IMRs puramente acústicos y composición de música original de su central impulsor Godfried-Willem Raes, principal fuente de consulta de este proyecto por poseer muy exhaustiva documentación de libre disponibilidad sobre la gran mayoría de sus creaciones.
- El vasto trabajo de Trimpin, en torno a instalaciones sonoras y dispositivos que incorporan mecatrónica, además de IMRs de muy preciso control parametrizable como el Contraption Instant Prepared Piano 71512 de los 80, diseñado para ejecutar las obras de Conlon Nancarrow para piano automático.
- La KarmetiK Machine Orchestra[5] (2010), co-dirigida por el especialista en tecnología aplicada a la música Ajay Kapur[6].
- El proyecto LEMUR[7] (2000) y la gran cantidad de desarrollos del músico e ingeniero Eric Singer[8], con instrumentos de muy alta capacidad expresiva como el GuitarBot.
- La Electric Motion Orchestra de Christof Schläger, con proyectos como Aural Shape y Urban Horns[9].
- El trabajo de Jacques Remus[10], creador de instalaciones y esculturas sonoras.
- Expressive Machines Musical Instruments (EMMI) de Troy Rogers, Scott Barton, y Steven Kemper de la Universidad e Virginia, EEUU.
- Las instalaciones sonoras Amorphic Robot Works, de Chico MacMurtrie[11]

4 https://logosfoundation.org
5 https://www.karmetik.com/karmetik-projects/the-machine-orchestra
6 https://www.ajaykapur.com
7 http://www.lemurbots.org/about.html
8 Quien fuera convocado en 2007 por Pat Metheny para la realización del proyecto Orchestrion. http://www.singerbots.com
9 https://christofschlaeger.de
10 http://jacques-remus.fr
11 http://amorphicrobotworks.org

Fig. 2: IMRs Modernos. De izquierda a derecha: KarmetiK Machine Orchestra en concierto, Pat Metheny junto al GuitarBot y otros orchestrions, e instrumentos de la Logos Robot Orchestra junto a Godfried-Willem Raes.

4. IMRs de percusión

Por otro lado, en cuanto a los instrumentos mecatrónicos específicamente percusivos, interés particular de este trabajo, han sido foco de investigación y servido de inspiración:

- El Notomoton[12], de KarmetiK (2010), consistente en un tambor con cuerpo metálico al que se le ensamblan 18 actuadores solenoides para alcanzar muy altas densidades cronométricas de ejecución.
- El Robotic Drums de Liat Segal[13] (2010), en el que 18 tambores Darbuka y 36 brazos robóticos se controlan mediante comunicación inalámbrica.
- El Glitch Robot del proyecto Sonic Robots interesante no solo por su sonoridad no convencional y su enfoque percusivo, sino también por utilizar diversos materiales recuperados para su confección.
- Y los instrumentos <Snar>[14], <Snar_2>[15] y <HAT>[16], de Godfried-Willem Raes, con amplia capacidad expresiva.

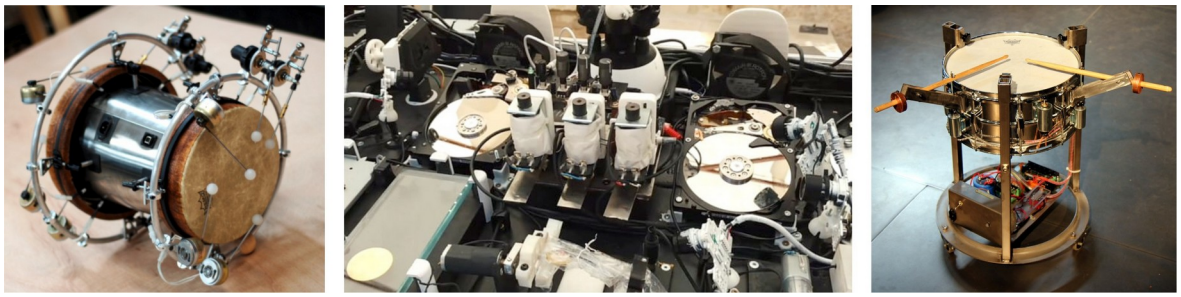

Fig. 3: NotomotoN (izq.), Glitch Robot (centro) y Snar_2 (der.)

12 https://www.karmetik.com/karmetik-robotics/notomoton/
13 http://www.liatsegal.com/2010/10/robotic-drums.html
14 https://www.logosfoundation.org/instrum_gwr/snar.html
15 https://www.logosfoundation.org/instrum_gwr/snar2.html
16 https://logosfoundation.org/instrum_gwr/HAT.html

5. IMRs en Argentina y América Latina

Es un aspecto pendiente para futuras investigaciones el indagar más exhaustivamente en torno a los desarrollos de IMRs y uso de mecatrónica en general para proyectos musicales en Argentina y América Latina. En una primera aproximación, se encuentran:

- Banda de robots[17], una iniciativa de talleres para niñxs, con un perfil lúdico y pedagógico, orientado a la construcción de pequeños instrumentos robóticos con dinámicas de trabajo colectivas.
- El trabajo de, Oliverio Duhalde[18] (impulsor del proyecto anterior), artista sonoro argentino que ha desarrollado esculturas e instalaciones sonoras, como KLOK y DANDELION, usando mecatrónica,
- La tesis de grado de Héctor Cruz, desarrollada en esta casa de estudios, consistente en un dispositivo mecatrónico de acción automática sobre el teclado de un órgano eléctrico italiano Giaccaglia Castelfidardo, de 1970.
- Diversos *instrumentos informales* introducidos por el grupo humorístico argentino Les Luthiers[19] como el "Campanófono a martillo" diseñado por Héctor Isamu, sobre una idea de Carlos Iraldi, en el cual una serie de martillos montados sobre solenoides rotativos accionan contra campanas tubulares. O el robot musical semi-antropomórfico "Antenor", también diseñado por Iraldi, con 13 cornetas con altavoces y tambores, el cual podía desplazarse por el escenario y hacer gestos faciales con un arreglo de LEDs ubicados en su cabeza giratoria.
- Trabajos desarrollados en centros de estudio de ingeniería y técnica como el *Sistema robótico para tocar una batería de música*, de Andrés Cela Rosero, Escuela Politécnica Nacional de Quito, Ecuador (Cela Rosero, 2007); o el *Robot educativo de bajo costo que interpreta melodías en piano* de Buitrago, Prieto y Peña, Facultad de Ingeniería, Universidad Pedagógica y Tecnológica de Colombia (Buitrago, Prieto y Peña, 2012)
- Variadas experiencias de utilización de robótica en el ámbito educativo, con alcances en torno a la enseñanza musical como el proyecto en torno al Robot Butiá en Uruguay (Benavides, Otegui, Aguirre, Andrade, 2013).

17 https://www.bandaderobots.com
18 https://www.oliverioduhalde.me
19 http://www.lesluthiers.com

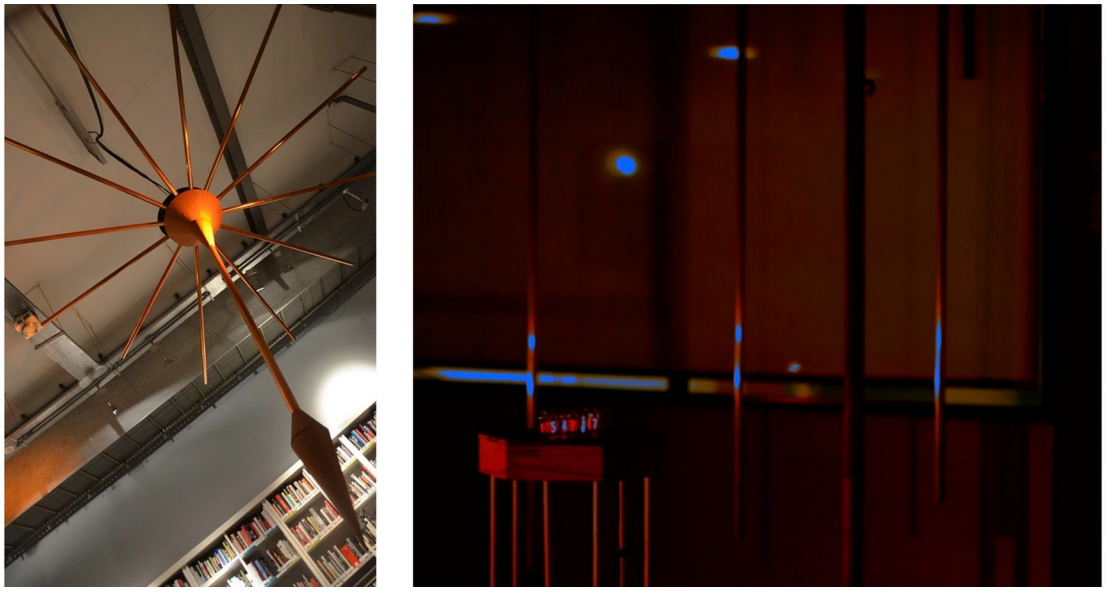

Fig. 4: Obras de Oliverio Duhalde usando mecatrónica: DANDELION (izq.) y KLOK (der.)

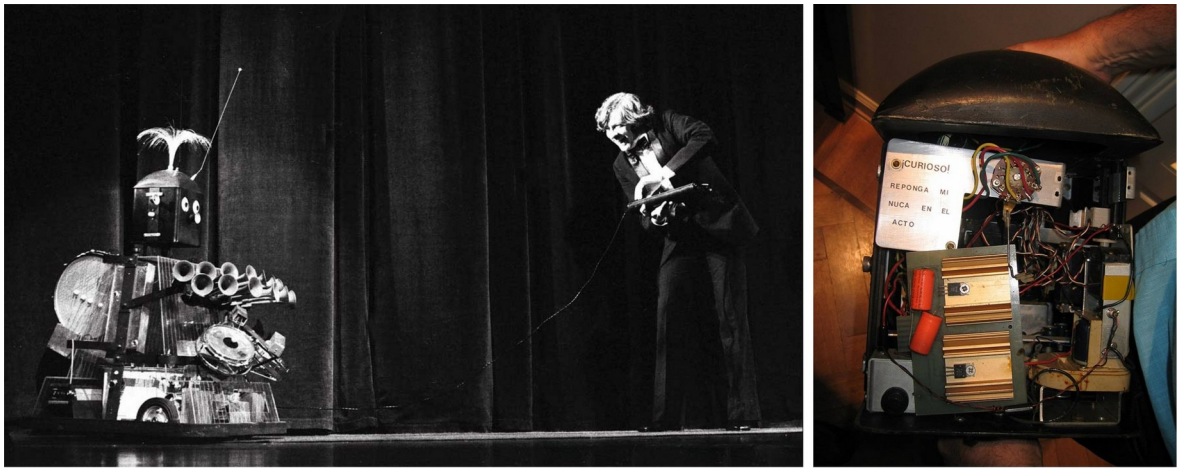

Fig. 5: Izquierda: Antenor de Les Luthiers, en escena junto a su "amo" Carlos López Puccio. Derecha: la cabeza de dicho robot desmontada, mostrando el humor del conjunto de instrumentos informales hasta en las "tripas" de sus creaciones.

6. Elementos conceptuales destacados en el desarrollo de IMRs

Según Maes, Raes y Rogers, los desarrollos de robótica orientada a música se pueden clasificar en: (1) *robots antropomórficos industriales*, los de mayor conocimiento público, diseñados como "*compañeros humanos y proveedores de servicios*", que ejecutan música para demostrar su destreza y avance tecnológico. Ejemplo de esto es el Toyota Partner Robot (Toyota Motor Corporation, 2003). A su vez, (2) robots producidos específicamente como *autómatas*

musicales, los cuales guardan rasgos antropomórficos en varios grados. Finalmente, (3) una creciente producción de *IMRs de naturaleza no-antropomórfica*, diseñados específicamente para excitar materiales y/o instrumentos, explotando al máximo las capacidades expresivas de dichas máquinas (Maes, Raes, Rogers, 2011).

Por su parte, Rogers, Kemper y Barton plantean que dentro del diverso campo de IMRs, podemos distinguir entre *(1) máquinas emulativas*, que ayudan a los desarrolladores a comprender y/o replicar a la ejecución músico-instrumental humana, y *(2) máquinas inventivas* desarrolladas por compositores y constructores que buscan nuevos vehículos para la expresión musical. (Rogers, Kemper, Barton, 2015).

En cuanto al diseño de IMRs, también Raes y equipo hacen hincapié en los siguientes aspectos: durabilidad, usabilidad, naturaleza acústica, capacidades y "legibilidad" de los autómatas (Raes, 1987-2020) (Raes, 1993) (Maes, Raes, Rogers, 2011).

- *Durabilidad*: enmarcándose en la tradición de construcción de instrumentos acústicos con longevidad en mente. Aquí la robustez, la utilización de materiales perdurables y la implementación de mecanismos electrónico-digitales y protocolos de comunicación extendidos, etc. juegan un papel central.
- *Usabilidad:* en cuanto a que el instrumento permita un uso accesible por compositores con mínima formación técnica. En este punto, Raes hace hincapié en hacer esfuerzos por implementar MIDI como protocolo de comunicación, al menos como una opción, a pesar de resultar limitado para las posibilidades expresivas de los IMRs actuales.
- *Naturaleza acústica:* para Raes, los IMRs de interés son los que excluyen el uso de parlantes, es decir aquellos puramente acústicos.
- *Capacidades*: apuntando a que los IMRs expandan las capacidades expresivas del ser humano, superando en algunos casos sus límites en la ejecución precisa.
- *"Legibilidad"*: que los autómatas estén "desnudos", es decir, que muestren su forma de construcción, tanto en la propia morfología del instrumento como en la documentación del artefacto.

Con puntos de contacto con lo anterior, para LEMUR (Eric Singer y equipo), el interés está en aumentar las posibilidades disponibles para los músicos y compositores y no en "*reemplazar a los músicos humanos*". Algunas de estas capacidades incluyen mayor velocidad de interpretación, granularidad de tono y expresión, ejecución de polirrítmicas complejas y reproducción de duración muy prolongada. Destacan que los IMRs proporcionan a los compositores una respuesta inmediata, similar a la de componer con sintetizadores. Pero que, a diferencia de los sintetizadores, los instrumentos físicos resuenan, proyectan e interactúan con los espacios sonoros de formas más ricas y complejas: "*claramente, también tienen una presencia física más dominante*".

También en palabras de dichos autores, los IMRs poseen una amplia variedad de aplicaciones, incluyendo instalaciones musicales automatizadas e interactivas, ejecución de obras musicales en directo (ya sea como instrumentos solistas, en combinaciones como un conjunto robótico, así como interactivamente junto con músicos humanos. (Singer, Feddersen, Redmon, Bowen, 2004).

Finalmente, en un debate que raya la filosofía y la praxis en música y arte contemporáneos, la motivación de Raes y Logos Foundation en la robótica y la generación de IMRs proviene de la opinión de que los altavoces, en tanto fuentes de sonido (una necesidad excluyente para cualquier sonido generado electrónicamente), son "*virtualizaciones de una realidad acústica*". Por tanto afirman, sin ahorrar en polémica, que estos "*tienden a socavar la razón de ser de los conciertos como rituales sociales*" (Raes, 1993) (Maes, Raes, Rogers, 2011) .

6.1. Aspectos de interés en el diseño de artefactos percusivos

En particular, el trabajo de Ajay Kapur y equipo ha sido de gran aporte a este proyecto, por su estudio comparativo sobre diseños de percutores para IMRs (Kapur, 2007) (Long, Murphy, Kapur, Carnegie, 2015) que ha servido como fuente de inspiración para el diseño de varios de los percutores aquí detallados, en particular los basados en solenoides lineales; su reporte pormenorizado sobre el mencionado instrumento NotomotoN (Kapur, Darling, Murphy, Hochenbaum, Diakopoulos, Trimpin, 2011), con el cual se observó el alto rendimiento de los solenoides rotativos; y su reporte sobre el Nudge (Murphy, Carnegie, Kapur, 2014), el cual mostró la potencia de combinar múltiples actuadores en el diseño de un mismo percutor.

Resultan de particular interés las nociones de estos autores para la evaluación de percutores basados en actuadores solenoides, electro-neumáticos y servomotores; varias de las cuales pueden servir también para estudiar otros tipos de IMRs:

- *Latencia:* el tiempo de respuesta global y por segmentos entre los componentes del IMR. Es decir desde la generación de la señal MIDI hasta su ingreso en el MC, de allí al driver de motor, luego al actuador en tanto estímulo eléctrico y finalmente al evento sonoro.
- *Sonoridad máxima* [a lo que se podría agregar mínima, y hablar de *rango dinámico*]: Sobre la intensidad sonora alcanzable por el dispositivo.
- *Consistencia Dinámica:* grado de fluctuación en la intensidad sonora frente a diversas tasas de repetición de la acción del percutor.
- *Máxima tasa de repetición:* máxima cantidad de acciones sobre unidad de tiempo alcanzables.

7. Trabajo con materiales tecnológicos recuperados

Finalmente, un breve comentario en torno a uno de los fundamentos del uso de chatarra tecnológica como materia prima principal en este proyecto: la problemática de la sociedad de consumo y su impacto socio-ambiental.

Según el informe "*A New Circular Vision for Electronics*"[20] de la ONU, para el Foro Económico Mundial de 2019, cada año se producen cerca de 50 millones de toneladas de residuos electrónicos y eléctricos. Esto equivale al peso de todos los aviones comerciales jamás construidos o a 4500 Torre Eiffel. De estos residuos altamente contaminantes, solo se recicla correctamente el 20%, pudiéndose recuperar prácticamente la totalidad de los mismos *(ONU, 2019).*

En un análisis de dicha problemática en general y sobre su impacto en la Argentina en particular, las investigadoras Peñaloza, Narvaez y Solanes exponen que el fenómeno de un constante consumo (compra e inminente desecho) de aparatos eléctricos y electrónicos (AEE), se

20 https://www.itu.int/en/ITU-D/Climate-Change/Documents/2019/A-New-Circular-Vision-for-Electronics.pdf

produce porque la industria tecnológica desarrolla nuevos aparatos que cubren más necesidades que los anteriores, haciendo que, aun cuando el artefacto en cuestión cuente con plena capacidad de uso, este ya deje de colmar las expectativas del usuario y se deseche. También señalan que muchos aparatos tienen una vida útil preestablecida por el fabricante, lo que hace que habiendo transcurrido cierta cantidad de tiempo, el aparato deba ser reemplazado nuevamente: lo que se conoce como "obsolescencia programada". En este sentido, vinculan esta problemática de la sociedad de consumo, no solo con un problema de desaprovechamiento de recursos sino también con una problemática ambiental preocupante, debido a que tanto la producción y el uso, como la posterior eliminación de los residuos de estos aparatos, requieren grandes cantidades de energía y la utilización de distintos tipos de materiales, que provocan contaminación, siendo perjudiciales para el medio ambiente y la salud.

A su vez agregan que, según datos de Greenpeace, a marzo de 2012, cada argentino genera alrededor de 3 kilogramos de basura electrónica por año, lo que representa 120 mil toneladas de basura electrónica anuales. Y que alrededor del 50% de estos residuos están arrumbados en oficinas, hogares, entes públicos o depósitos, más del 40% se entierra o se descarta en basurales y rellenos y solo un 10% ingresa en esquemas informales o formales de gestión de residuos. Todo lo cual representa un derroche de recursos que podrían recuperarse, además de una alta fuente de contaminación (Peñaloza, Narvaez y Solanes, 2014).

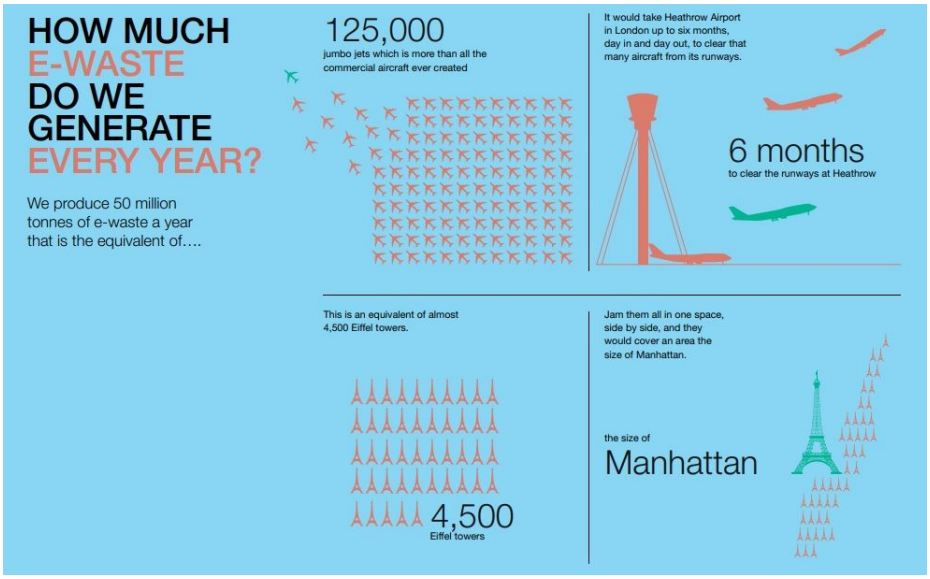

Fig. 6: Infografía del mencionado documento de ONU sobre los equivalentes de la chatarra tecnológica generada en un año.

Finalmente, en el mencionado escrito de Naciones Unidas se plantea que "s*e necesita una nueva visión para la producción y el consumo de productos electrónicos y eléctricos. Es fácil sesgar el problema unicamente en el consumidor, pero el asunto es mucho más amplio. Los fabricantes, inversores, comerciantes, mineros, productores de materias primas, consumidores, responsables políticos y otros tienen un papel crucial que desempeñar en la reducción y reciclado de estos residuos." (ONU, 2019).*

Desarrollo

1. Recuperación de materiales y trabajo con chatarra tecnológica

1.1. Consecución de chatarra tecnológica

Si bien se contaba con un cantidad base de materia prima para previo al inicio del proyecto, la consecución del grueso del material se logró a partir de del lanzamiento de una campaña de difusión, pidiendo donaciones de chatarra tecnológica.

Fig. 7: Flyer difusión campaña recolección de chatarra tecnológica

Dicha campaña constaba inicialmente de 2 partes, la primera orientada al entorno cercano de quien suscribe y la segunda, focalizada en industrias, organismos estatales y entidades similares.

La misma tuvo una excelente recepción en relación a su proyección inicial. Se alcanzó a unas 150 - 200 personas, y se consiguieron:

- 27 impresoras domésticas tipo chorro tinta de marcas varias
- 2 fotocopiadoras de mediana escala
- 1 fotocopiadora profesional

- Varios artefactos más como ser televisores, equipos de audio, parlantes, cables, etc.

Dado que la primer etapa tuvo un gran éxito y se cubrieron con creces los requerimientos establecidos para el proyecto, no hubo necesidad de llevar a cabo la segunda etapa.

1.2. Desarme y extracción de materiales

Aspecto para nada desestimable en el trabajo con chatarra tecnológica es el momento del desarme de artefactos y la recuperación del material útil. Si bien en líneas generales no se identificó necesidad de contar con herramientas especiales, más allá de destornilladores, pinzas, llaves, el tiempo y esfuerzo que demanda esta tarea es importante e impacta de manera considerable en proyectos de gran escala. Término medio una impresora doméstica tipo chorro a tinta demanda entre 20 minutos y media hora, desde que comienza el desarme hasta la extracción y preparación para el descarte del material residual. Por otro lado, una fotocopiadora de gran escala puede demandar entre 3 y 6 horas.

Dentro del procedimiento, algo a destacar es la importancia de no forzar o romper el material para la extracción de componentes útiles. Esto puede traer como contrapartida el daño irreparable de los componentes que se quiera recuperar[21]

En el mismo sentido el espacio necesario para el almacenamiento de los artefactos, desde su consecución hasta su desarme para extracción de componentes útiles es una dimensión a considerar en los proyectos que utilicen esta fuente de material.

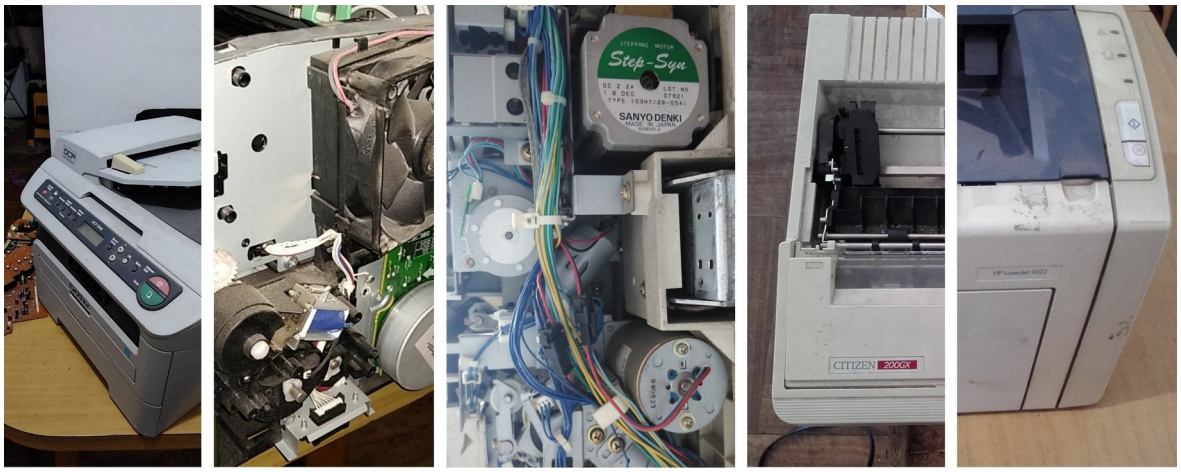

Fig. 8: Registro de recolección y desarme de chatarra tecnológica

1.3. Recuperación de motores, componentes electrónicos y material estructural

De todo el lote de artefactos conseguidos se extrajeron un total de 76 motores de diversa naturaleza y tamaño.

21 Es común este problema en artefactos en los que se desconoce su arquitectura interna, que al intentar romperlo para acelerar el proceso de extracción se termina dañando parte de material útil del cual se desconoce su ubicación en el aparato.

- 31 motores Paso a Paso
- 28 Motores rotativos de corriente continua
- 17 solenoides lineales

Fig. 9: Motores recuperados

Esto, además de material estructural de los artefactos como metal, plástico, tornillos, tuercas, etc. Así también, cables de diversos grosores, y varios componentes electrónicos de valor.

Fig. 10: Material estructural recuperado

1.4. Recuperación de Batería

Durante la campaña de recolección de materiales también se consiguió ademas 1 batería acústica, aunque en mal estado, por lo que debió ser restaurada.

Fig. 11: Proceso de restauración de tambores de batería recuperada

1.5. Conclusiones en torno al trabajo con materiales recuperados

Se trata de un material que no suele tener una respuesta lineal o común en todos los objetos del mismo tipo. Mismos modelos de motor o componentes devuelven performances distintas dado su diferente grado de desgaste, poco estimable a priori, a partir de su uso. A su vez es un material del que difícilmente se encuentren especificaciones técnicas.

Esto es particularmente notable en el punto más sensible del sistema: los motores. El voltaje de trabajo, corriente máxima, torque de empuje, de retención, etc. no son valores con los que, por lo general, se cuente a priori. En muchos casos estas propiedades se deben estimar, en un procedimiento empírico de "aproximación exógena".

Esto plantea una forma particular de encarar el trabajo de construcción: en definitiva los diseños deben partir específicamente tomar como base esas instancias de material, esos motores y componentes. No se trata de un material en abstracto, este tiene rasgos particulares, una historia, fallas, etc. Todo esto no solo repercute en las condiciones de construcción y características de funcionamiento sino también en el aspecto estético del sistema. De aquí que cada diseño porta de manera indisimulable, la huella de su origen.

2. Control lógico

Se tratará en este capítulo lo referente a la disposición y programación de los microcontroladores utilizados en el proyecto. Se trata de un sistema embebido, el cual tiene como propósito central traducir mensajes de control MIDI (Note On / Off y Control Change) en señales eléctricas para control de drivers de 3 tipos de motores: solenoides lineales o motores rotativos de corriente continua (con funcionalidad equivalente), paso a paso (PAP), y servomotores. A su vez, el programa provee de manera sencilla mecanismos de monitoreo de estado de ejecución del programa, como ser indicadores LED y mensajes por puerto serie para la fase desarrollo y corrección de errores.

Tanto en su fase prototípica como definitiva, dicha programación se mueve en el universo de desarrollo de sistemas de Open Hardware y creación de proyectos DIY Arduino y similares. Así mismo, en cuanto a software de terceros, en ambos momentos del trabajo se utilizaron las siguientes librerías: Arduino MIDI Library de FortySevenEffects[22], la Servo library desarrollada por Arduino, AccelStepper de Mike McCauley[23] y ESP32Servo[24].

En los anexos de este documento se adjunta la transcripción completa del código.

2.1. Programación en fase prototípica

La programación en esta primera fase se realizó utilizando lenguaje Arduino básico (basado en C/C++), implementando un paradigma de programación procedural. Los modelos de Arduino utilizados fueron Duemilanove, UNO y Mega2560. Se utilizó la IDE provista por Arduino en sistema operativo Windows 8.1. Es interesante mencionar también que se utilizó como mecanismo de archivo y control de versiones (artesanal) la herramienta Google Drive.

El código fue estructurado en 4 archivos:

- ***PulsAr.ino:*** Archivo principal el cual cuenta con las funciones núcleo del programa - setup y loop-. Aquí se invocan las librerías utilizadas y se crean los objetos tipo MIDI_INTERFACE y Servo. Se disparan el conjunto de las funciones cíclicas del programa.
- ***midi.ino:*** contiene las funciones que intermedian entre la lectura de la interfaz MIDI y la invocación a los procedimientos de generación de señales de control de motores.
- ***pap.ino:*** Contiene las funciones que generan las señales de control de los drivers de motor tipo PAP.
- ***solenoides.ino:*** Contiene las funciones que generan las señales de control de los drivers de motor tipo solenoide.

22 https://github.com/FortySevenEffects/arduino_midi_library. Versión 4.3.1. Cabe destacar que el proyecto utiliza una versión anterior a la más recientemente publicada (5.0.2), dado que el cambio de versión ocurrió durante la finalización del proyecto.

23 https://www.airspayce.com/mikem/arduino/AccelStepper/ Versión 1.61. 2010-2018.

24 Versión 0.9.0

2.1.1 Estructura del programa

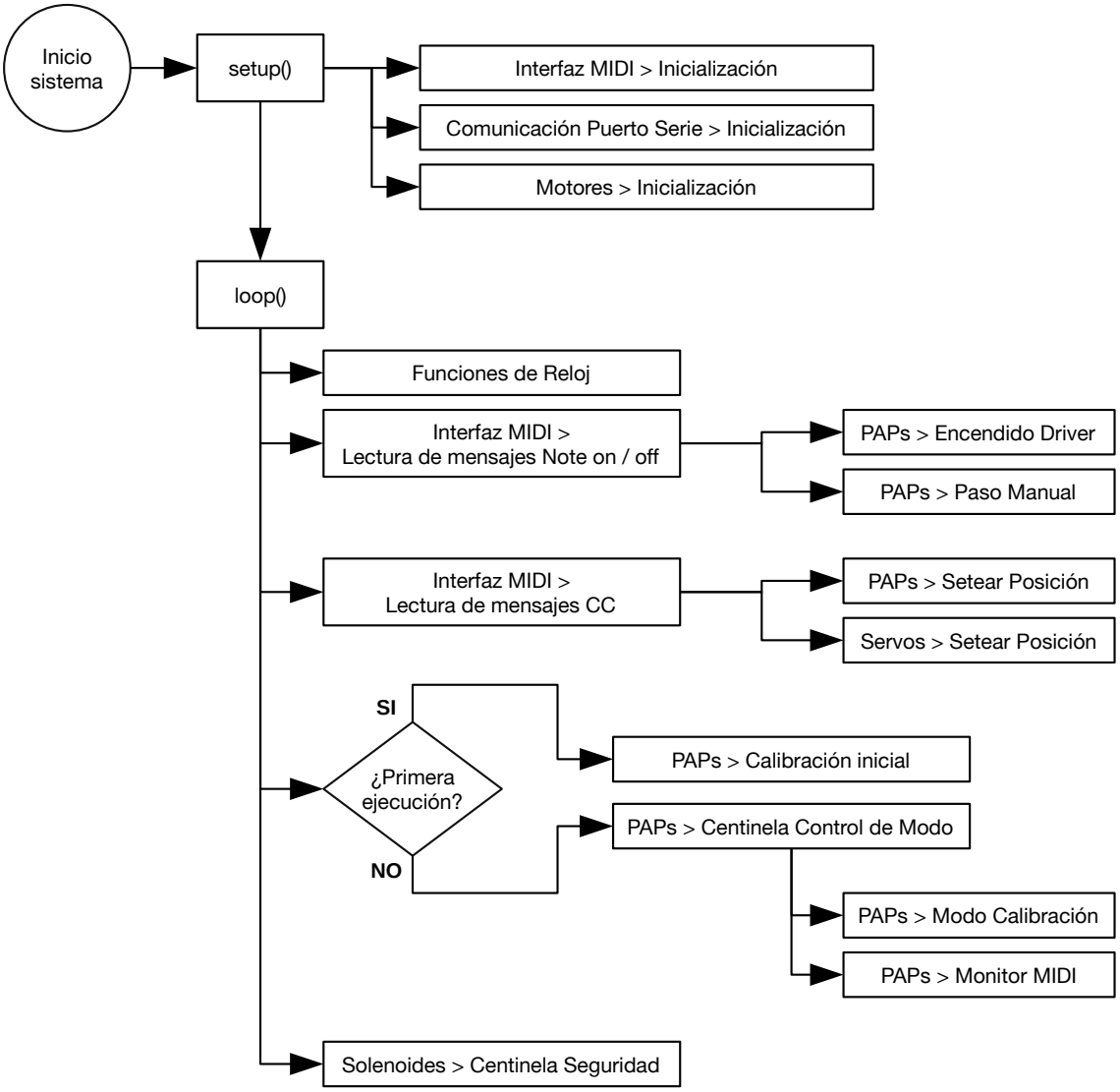

Fig. 12: Funcionamiento principal programa MC versión prototipo

2.1.2 Funciones de reloj

Desde el comienzo de la programación se evitó utilizar interrupciones del procesador (función `delay()` del lenguaje Arduino), lo cual impide un eficiente y plenamente responsivo funcionamiento en paralelo de múltiples dispositivos controlados por el mismo procesador. El

control temporal de las duraciones de los eventos, se realizó mediante el registro y comparación de la ocurrencia de los diferentes eventos basado en el valor retornado por la función `micros()`[25].

2.1.3 Procedimientos de control y seguridad de motores solenoide

El mecanismo de control de solenoides consistió en la traducción de mensajes MIDI de Note On / Off a una señal PWM escalada de manera proporcional al valor de velocidad (*key velocity*) de la nota accionada. Dicha señal eléctrica, apunta a controlar un driver simple de motor de corriente continua. El ancho de pulso diferencial de la señal de control busca traducirse en modificaciones en la velocidad del movimiento resultante en el motor controlado y, ulteriormente, en la posibilidad de escalar la intensidad sonora[26].

Por otro lado, a fin de proteger los drivers, solenoides y la fuente de alimentación eléctrica, se incluyó un procedimiento de limitación de la duración máxima de la acción del solenoide. Dicho procedimiento consiste en registrar el momento de inicio de la acción del motor, mediante valores de reloj, para luego, en cada ciclo de `loop()`, en caso de encontrar un solenoide accionado, se compara el tiempo transcurrido entre el inicio de la acción y el momento actual y, en caso de superar un límite máximo (establecido por defecto en 1.5 segundos) el sistema fuerza el apagado del solenoide.

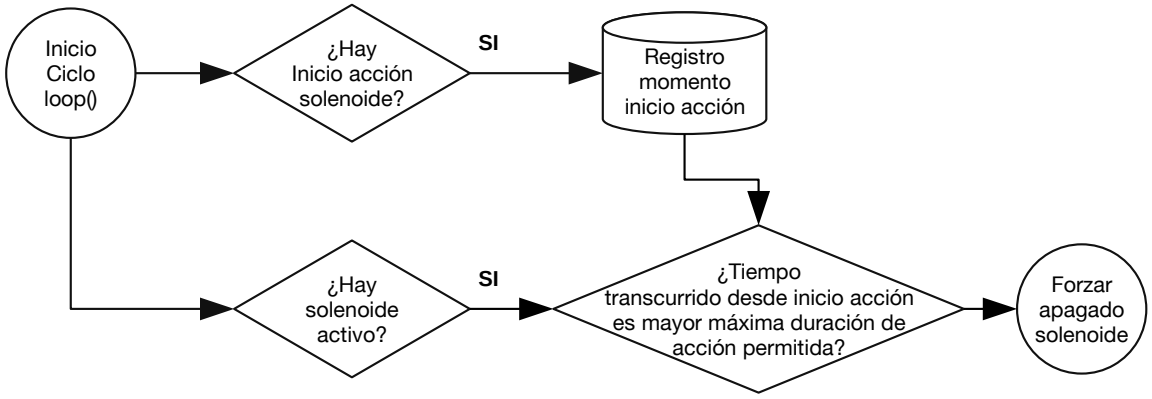

Fig. 13: Flujo de procedimiento de seguridad de solenoide

25 Cabe mencionar que en las primeras versiones del programa, se registraba el valor de `micros()` en una variable para evitar hacer múltiples ejecuciones de dicha función en un mismo ciclo de `loop()`. Esto se mostró como una estrategia errónea dado que entre la ejecución de las diferentes instrucciones de dicho ciclo de `loop()` el MC registra nuevos valores de reloj, por lo que el registrarlo una vez y reutilizar ese mismo valor redunda en una pérdida de resolución del control temporal y, por tanto, en una programación errónea.

26 Como se menciona posteriormente, este mecanismo será evitado en el desarrollo definitivo, dados los grandes niveles de generación de artefactos sonoros indeseable indeseables, producidos por vibraciones resonantes en el cuerpo de los motores, sobre todo en los rotativos de corriente continua.

2.1.4 Procedimientos de Control de motores PAP

El programa para control de motores PAP tuvo por objetivo la traducción de mensajes MIDI de cambio de control[27] en cambios en la posición del motor. Los valores 0-127 de CC MIDI se mapearon a una posición mínima y máxima de giro del actuador.

Se utilizaron motores PAP en 2 tipos de actuadores: agitadores de granos (shakers) y percutores. Es así, que la programación orientada hacia el control de este tipo de motores debió resolver tanto el movimiento rotativo de los mismos como, en el caso del percutor de tambor, su calibración, incluyendo mecanismos de detección de posición inicial mediante sistema de final de carrera[28].

La mayor complejidad de construcción y uso de los motores PAP respecto a los solenoides lineales o rotativos, se traduce en mayor complejidad en el driver a utilizar. Por tanto la programación orientada al manejo de los mismos son también más desafiantes. Mientras que de cara al motor solenoide lineal debe simplemente manejarse el flujo de corriente hacia una única bobina, sin necesidad de controlar la polaridad / dirección del flujo de dicha corriente, los motores PAP bipolares[29] poseen 2 bobinas sobre las que se debe controlar tanto la presencia de corriente como el flujo de polaridad.

Luego de varias pruebas con driver de motor PAP basadas en circuito *Doble Puente H* (tanto con componentes discretos como con integrado L298N[30]), se resolvió usar los drivers símil Pololu drv8825 o a4988, comúnmente utilizados en impresoras 3D del proyecto Rep Rap, ambos pin-compatibles entre sí.

El uso de dichos drivers simplifica enormemente la programación a construir dado que la misma se resume en la generación controlada de un tren de pulsos para la realización de los pasos, y una señal aún más sencilla para el control binario de dirección de paso (horario o antihorario). Cabe mencionar que también se reduce la cantidad de pines de salida a utilizar en el MC por cada motor, pasando de al menos 4 usando *Doble Puente H*, a 2 pines, en los drv8825 o símil[31]. Así también, dichos drivers permiten la utilización de la técnica de micro-paso, la cual permite un rendimiento superior del actuador[3233]. A pesar de que la electrónica necesaria para el

27 *Control change*, a partir de ahora CC.

28 En adelante, FdC. Se trata de un interruptor eléctrico instalado en un extremo del recorrido completo del actuador, contrario a la dirección del objeto a percutir, cuya función es detectar que una parte mecánica del dispositivo ha entrado en contacto con el mismo y por tanto ha llegado al "final de la carrera".

29 Diseño más simple de controlar, el cual es el utilizado en este proyecto

30 Detallado en el apartado Circuitos electrónicos sobre Drivers de Motor PAP

31 3 si se quiere incluir un control de activación del driver.

32 En vez de polarizar sendas bobinas con nula o máxima tensión, la técnica de micropaso implica establecer un escalado fraccional de la tensión entre +VCC y -VCC y un desfase de 90º entre las señales de alimentación de las bobinas. Esta técnica posibilita aumentar la resolución de giro del rotor, incrementando la cantidad de pasos por vuelta y reduce las vibraciones y por tanto el ruido sonoro emitido por el actuador, entre otras ventajas.

33 Cabe mencionar sin embargo, que el uso de esta técnica vuelve al motor más propenso a la llamada "pérdida de pasos". Este es un fenómeno en el que un motor PAP, o bien no puede completar un paso en la dirección de giro solicitada (el cual puede producirse por una resistencia no contrarrestable de un objeto que se opone al giro del rotor, una falla por desgaste del mismo, un salto drástico en la velocidad de giro requerida, etc.) o bien es modificada su posición en una situación de retención por una fuerza aplicada exteriormente. Dado que el motor PAP no posee un mecanismo de retroalimentación (*feedback*), del tipo codificador (*encoder*) o similar, que permita corroborar si se ha efectuado el giro o se ha mantenido la posición según lo requerido, se produce un desfasaje entre el registro del contador de pasos del MC y los pasos efectuados realmente por el actuador. En resumen, el dispositivo se descalibra por una pérdida los pasos contados.

empleo de este mecanismo es bastante más compleja, los drv8825 o símil, permiten resolver la implementación de manera muy sencilla, transformándose en una *solución-todo-en-uno*.

2.1.4.1 Control de pasos y dirección

El control de pasos consiste en la generación un tren de pulsos[34] con forma de onda cuadrada (pico y valle simétricos) con frecuencia modificable en un pin de salida. La frecuencia de dicho TDP impactará en la velocidad de rotación del motor. Los pulsos necesarios para esta señal se generan siguiendo estos pasos:

1. Se guarda en una variable un valor ***dsc*** correspondiente a la duración en microsegundos del semiciclo del tren de pulsos.
2. Se registra el momento de reloj en que inicia la escritura de voltaje alto ***iva*** en el pin en cuestión
3. Se corrobora en cada ciclo de `loop()` si el tiempo transcurrido desde ***iva*** es igual o mayor a ***dsc***. En caso de que si, se escribe un voltaje bajo (LOW) en el pin de salida y se hace un proceso simétrico pero con este nuevo valor de tensión. El resultante será una onda cuadrada.

Por otro lado, el control de dirección de giro del motor se produce mediante la presencia o no de tensión en el pin correspondiente.

2.1.4.2 Control de posición del rotor

El control del ángulo de giro del rotor se obtiene mediante el mapeo de mensajes MIDI de CC en el recorrido angular del rotor. Para ello:

1. Se guarda una variable de posición actual del percutor ***ppa***, medida en pasos desde el final de carrera
2. Se obtiene de la interfaz MIDI la lectura de mensajes de cambio de control (*control change*) correspondiente al percutor en cuestión. Dicho valor establece la posición requerida ***ppr*** del percutor.
3. En cada ciclo de procesador, se compara si ***ppa*** y ***ppr*** son iguales. En caso de no serlo:
   a. se calcula la dirección en que hay realizar pasos en el motor para ir en busca de igualar ***ppa*** y ***ppr***,
   b. se llaman a los procedimientos de ejecución de pasos.
   c. Al completarse el paso se registra la nueva ***ppa*** y se repite el proceso

2.1.4.3 Programa para agitador (shaker) basado en PAP

Los procedimientos hasta aquí descriptos son suficientes para el agitador de granos basado en PAP. El mismo obtiene un buen rendimiento y permite prescindir de la necesidad de establecer una posición concreta y recorrido angular de giro, dado que de lo que aquí se trata es de "recorrer distancias a secas", en diferentes velocidades, en la trayectoria circular cíclica del motor, sin importar las ubicaciones precisas en el espacio. Esto cambia drásticamente el percutor basado en PAP, aumentando muy significativamente la complejidad del programa.

34 TDP. Alternancia de un voltaje de control encendido (HIGH, correspondiente a 3.3 o 5 voltios de acuerdo a la tensión de trabajo del MC) y apagado (0 voltios) en el pin correspondiente.

2.1.4.4 Programa para percutor basado en PAP

El percutor basado en motor PAP incorpora el mapeo de mensajes MIDI de CC en un recorrido angular del rotor, tal como se ha descripto anteriormente, pero requiere de acotar el angulo de giro y establecer posiciones concretas de inicio y fin del giro útil, o dicho de otra forma posiciones mínima y máxima. En función de esto, se incluye el mencionado mecanismo de FdC: ya que es necesario que el MC conozca 3 posiciones clave en todo el recorrido angular, desde el FdC hasta el objetivo a impactar. Estas son:

1. *Posición de final de carrera (PFC):* es el "kilómetro 0", el punto de origen para la medición del resto de las posiciones.
2. *Posición de en-descanso (PED)*: posición en la que el percutor iniciará su recorrido útil en los movimientos para alcanzar el objetivo a impactar, la cual muy probablemente no coincida con la PFC. Esta es la posición útil del percutor más alejada del cuerpo a excitar.
3. *Posición de objetivo -a impactar- (PO)*: posición máxima de avance del percutor, la cual coincidirá con el punto de máxima compresión en / penetración del objeto a percutir, y a partir de la cual, de requerir mayor avance, el motor comienza a "perder pasos" de manera significativa.

A fin de registrar estas posiciones, se requiere un mecanismo de calibración. Para ello, por un lado, se incorporan 3 controles de NoteOn desde la interfaz MIDI para disparar procedimientos de: *Incrementar Paso*, *Decrementar Paso* y *Guardar posición / Re-calibrar*. Por otro lado, se establecen 2 modos de operación del programa: *A. Calibración* y *B. Uso*. A su vez, el modo calibración cuenta con dos submodos: *A.1. Calibración Inicial*, para el momento inmediatamente posterior al encendido del MC y de uso por única vez, y *A.2. Re-calibración*, para utilizarse posteriormente, toda vez que se requiera.

Ambos submodos de calibración son similares: los dos ejecutan un procedimiento de *calibración de final de carrera*; pero sólo el primero incluye procedimientos de *calibración de en-descanso* y *calibración de posición de objetivo*.

Entonces, la *Calibración inicial* posee tres momentos. En primer lugar, la *calibración de final de carrera*, hace girar el percutor hasta el mecanismo de FdC. Al detectarse voltaje alto en el pin de entrada correspondiente a FdC, se constata la posición exacta del brazo del percutor y se puede, a partir de allí, controlar de manera precisa su posición posterior. Esto, tomando como posición 0 la ubicación del percutor al momento de hacer contacto con el FdC y contando a partir de allí los pasos dados en dirección al objeto a impactar.

En segundo lugar, se procede a la *calibración de en-descanso*. Siendo que el sistema no dispone de sensores para determinar de manera automática la distancia hasta el objetivo de impacto, se deben realizar los pasos de calibración restantes de forma asistida. El usuario deberá accionar *Incrementar Paso*, para llevar el percutor hasta la PED deseada. Una vez realizado esto, se deberá accionar *Guardar Posición*, para que el sistema registre PED.

Finalmente, en tercer lugar, se ejecuta la *calibración de posición de objetivo de impacto*, la cual determina a qué cantidad de pasos se encuentra el PO del FdC. Esta es equivalente a la anterior en términos de procedimiento, solo que su efecto es muy distinto. Cabe aclarar que una vez guardada la tercera posicisión clave (PO), el sistema bloquea el acceso a este modo de calibración inicial.

Finalizada la calibración, se pasa a *Modo Uso*. El mismo comparte el mecanismo de uso antes descripto (mediante CC MIDI), pero incorpora la posibilidad de volver a efectuar la *calibración de final de carrera*. Esto, a fin de salvar la eventual pérdida de pasos en la operación del actuador. Esta se hace automáticamente, al volver a ejecutar un NoteOn correspondiente a *Guardar posición / Re-calibrar.*

2.1.5 Resultados obtenidos en fase prototipo

En este estadio, el programa cumplió satisfactoriamente con las necesidades de testeo preliminar de los instrumentos construidos, en sus correspondientes fases de prototipado. Cabe mencionar igualmente que en las pruebas realizadas con la placa Arduino MEGA, se observó que con el aumento de cantidad de ingreso de mensajes MIDI, se producía una pérdida de rendimiento (fluctuaciones en la frecuencia) en la generación de los pulsos de control para los drivers de motores PAP. En parte, este resultado motivó el cambio de la placa de desarrollo a una con mayor capacidad de procesamiento, que pudiera minimizar las eventuales fluctuaciones del sistema, concentrándolas lo más posible en el ámbito electro-mecánico.

Sin embargo, se observaron una serie de problemas en lo que hace al proceso de desarrollo y mantenimiento del código, típicos en la implementación de una estructura de programación procedural en proyectos que superan un nivel de complejidad bajo.

Entre los problemas identificados se encuentran:

- La arquitectura de la programación poseía una imbricación casi total entre sus partes; es decir, la implementación MIDI, la generación de señales de control, el monitoreo por puerto Serie, etc. se entrelaza en el código, sin poder distinguir con claridad los bloques dedicados a cada aspecto. Esto, además de dificultar el mantenimiento, vuelve complejo el reemplazo de un bloque, de una librería, etc. ya que impacta en el conjunto del desarrollo.
- Conforme el proyecto fue creciendo, la cantidad de variables a definir en cada ámbito del desarrollo volvía confuso el código.
- Cabe señalar que los problemas mencionados impactan a su vez negativamente en la posibilidad de reutilización de este código por terceros y, por tanto, con uno de los objetivos del proyecto en su conjunto, consistente en la contribución a la comunidad DIY mediante la socialización de los desarrollos desde una perspectiva de software y hardware abierto.
- A su vez, la estructura de "apilado" e iteración de las entidades creadas en el código se basó en arrays multidimensionales, los cuales resultan en una aproximación poco dúctil para escalar la cantidad de dichas unidades.
- Por otro lado, se observó que el enfoque con el que se elaboró el control de acción de motores PAP, mediante variaciones de Control Change MIDI, propició dificultades para la secuenciación MIDI y más aún para la interpretación en tiempo real con un controlador.
- Finalmente, el sistema de control de versiones era, por supuesto, muy rudimentario y artesanal, con dificultad para un correcto control y registro de cambios. En sintonía, la IDE utilizada se mostró limitada para el flujo de trabajo buscado.

Sobre esta base, se orientó la fase definitiva del desarrollo, la cual implicaría la superación del paradigma procedural de programación, teniendo en mente el alcanzar mayor independencia entre sus partes, es decir mayor modularidad, posibilidad de reutilización, etc. En el mismo sentido, se buscaría avanzar en una mejora del entorno de trabajo utilizado.

2.2. Programación en fase definitiva

La versión definitiva del programa del MC se construyó fundamentalmente utilizando programación orientada a objetos de C++. A su vez, se buscó una arquitectura más modular enfocada en que las distintas partes del código tuvieran independencia una de otra, pudiendo ser modificadas más fácilmente[35].

Por otro lado, se modificó la placa de desarrollo a fin de contar con mayor versatilidad y velocidad de procesamiento, dejando atrás el Arduino Mega 2560, avanzando a utilizar Nodemcu Espressif ESP32.

A su vez, a partir del uso de POO, junto la nueva placa de desarrollo, se debió hacer una exploración mayor en la lógica y sintaxis de C++, incorporando estructuras como vectores (arrays con tamaño variable modificable dinámicamente), etc.

Además, se modificó la IDE utilizada en el trabajo, cambiando la IDE de Arduino por el editor de código Open Source Atom[36], junto con el complemento PlatformIO, ecosistema orientado a sistemas embebidos. El nuevo programa se planteó como proyecto de PlataformIO orientado a Esspresif ESP32 con framework de Arduino.

Dentro del flujo de trabajo se implementó Git como repositorio de código / control de versiones y GitLab como servidor Git[37]. Finalmente, cabe mencionar que toda la fase final de trabajo se desarrolló utilizando sistemas basados en Linux (Ubuntu 18.4 y Lubuntu 19.10).

2.2.1 Dos versiones: v2 y v3

En el proceso de desarrollo definitivo se realizaron 2 versiones, v2 y v3, con 2 diferencias fundamentales entre si:

- La arquitectura más modularizada empleada inicialmente en v2 redundaba en el requerimiento de una gran cantidad de procedimientos de interpretación y procesamiento de datos. Esto planteó una proyección que se visualizó como un desequilibrio potencialmente ineficiente e indeseable para un sistema embebido, donde el límite de memoria y capacidad de proceso es más cercano que por ejemplo en una aplicación web. Más aún, tratándose de un sistema orientado a la producción de música, lo cual pone como prioridad insoslayable mantener la latencia en los tiempos de respuesta de los procedimientos cruciales de funcionamiento lo suficientemente baja como para experimentar una percepción psico-acústico-motriz lo más cercana posible al “tiempo

35 La búsqueda de una menor imbricación y mayor modularidad entre los segmentos de la programación se inspira en la idea de “capas de abstracción”, un paradigma de desarrollo de sistemas en el que la interconexión entre las partes del mismo se construye “apuntando a universales”. Es que las partes de dicho sistema no esperan de (ni, por tanto, se orientan a) una contraparte específica con una arquitectura predefinida. Por el contrario, se presupone como destino un ente lo más abstracto y genérico posible.

36 https://atom.io/

37 Cabe mencionar, como se inscribe al final del documento que el programa completo desarrollado se encuentra disponible en GitHub, servidor Git mayor difusión en la comunidad Open Source y DIY.

real". Esto fue una razón de peso para abandonar la estrategia planteada en v2, avanzando a v3, mudando los aspectos más funcionales de su predecesora.

- Por otro lado, conforme se realizaban pruebas de funcionamiento y estrés del MC con v2, se constató la necesidad de repartir las operaciones de control lógico en 2 unidades MC descentralizadas[38]. Esto desde ya modificó los aspectos de más alto nivel del programa, pero se buscó dejar intacto el resto del desarrollo, terminando en v3 con un solo código, compilable en sendos MC, con unas mínimas variaciones de configuración. Esto se documenta con mayor detalle al final de este apartado.

2.2.2 Programa v2

El código se estructuró en 7 archivos de C/C++:

- **main.cpp:** Archivo principal el cual cuenta con las funciones núcleo del programa -setup y loop-. Se ingresa la configuración inicial de cantidad de actuadores, mensajes MIDI de control y puertos de salida, variables de configuración general del sistema, etc. Se instancian los objetos GestorMIDI y ControladorActuadores.
- **GestorMIDI.h:** Correspondiente a la clase homónima. La misma se encarga de registrar en un vector (registroMIDI) los datos de entrada de la interfaz MIDI, provistos por la librería MIDI ya mencionada.
- **ControladorActuadores.h:** Correspondiente a la clase homónima, la cual se encarga de instanciar las clases de control de actuadores, y de pasarles la información del Registro MIDI, para que éstas resuelvan la elaboración de las señales eléctricas de control de motores.
- **Actuador.h:** Correspondiente a la clase homónima. Contiene una clase abstracta que estructura la forma general de las subclases específicas de tipo de motor.
- **Solenoide.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de motores solenoides
- **PAP.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de motores PAP
- **Servo.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de servomotores.

En líneas generales las estrategias de abordaje elementales para la creación de las señales eléctricas de control de motores son las mismas en esta fase de desarrollo que en la de prototipado. En este aspecto, además de las diferencias globales ya citadas, entre los puntos más contrastantes entre ambas fases encontramos:

- Separación completa del ámbito de recepción de mensajes MIDI del de lectura. Antes, el propio proceso de lectura disparaba las funciones de acción de motores. El nuevo programa simplemente guarda estos mensajes en un registro, sin "importarle" lo que suceda con el resto del programa.
- El objeto clase ControladorActuadores realiza el trabajo de unir el momento MIDI con el momento Actuadores[39].

38 Fundamentalmente al aunar control de servos, con motores PAP y solenoides lineales.

- Actuadores basados en PAP:
  - Eliminación del percutor PAP con final de carrera.
  - Modificación del tipo de mensaje MIDI usado para el control de los agitadores (shakers), pasando de CC a Note On / Off. La dirección de giro del motor está dada por la nota ingresada y su siguiente, siendo la primera en sentido horario y la seguna su reverso.

2.2.3 Programa v3: versión final

Esta es la versión definitiva del programa en cuanto a estrategia general para las tareas requeridas del MC. Se estructura en 5 archivos:

- **main.cpp:** Archivo principal el cual cuenta con las funciones núcleo del programa -setup y loop-. Se ingresa la configuración inicial del sistema (cantidad de actuadores por cada tipo y sus correspondientes mensajes MIDI de control y puertos de salida), variables de configuración general del sistema (como modo desarrollo), etc. Se instancia MidiInterface y PulsArControlador y se incluyen procedimientos auxiliares de callback[40] para vinculación de ambas instancias.
- **PulsArControlador.h:** contiene la clase homónima, la cual instancia las clases de control ulterior de actuadores y distribuye los mensajes MIDI según corresponda.
- **PulsArSolenoide.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de motores solenoides
- **PulsArAgitPAP.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de motores PAP
- **PulsArServo.h:** Correspondiente a la clase homónima, encargada de generar las señales eléctricas de control de servomotores.

2.2.3.1 Reorganización de la estructura de microcontroladores

Como se ha dicho, en este punto se resolvió utilizar 2 unidades microcontroladoras para distribuir de manera más eficiente las tareas de control lógico. Se indagaron 2 formas de organizar los MC:

- Sistema centralizado: Una placa MC madre que sea la única que reciba los mensajes MIDI, los procese y además de generar las propias señales de control para actuadores, luego envíe señales digitales de control (por puerto serie) utilizando algún protocolo de comunicación a una segunda MC.
- Sistema descentralizado: Dividir el sistema en partes controladas individualmente, donde todas tengan posibilidad de recepción, procesamiento (y eventual reenvío) de mensajes MIDI. De implementarse este sistema, se deberá evaluar a su vez la mejor forma de interconexionado, pudiendo utilizarse una tipología tipo encadenado o línea, o bien tipo estrella, usando un MIDI Splitter.

39 Serán estas 2 modificaciones las que se discontinuarán en v3, encontrando una versión superadora que permita aprovechar los procedimientos de callback (ver nota siguiente) disponibles en la librería MIDI, perdiendo en un mayor grado de imbricación entre las partes del código, pero mejorando notablemente la eficiencia del programa.

40 "Retrollamada" que realiza una función o procedimiento A a una función o procedimiento B, que se pasa como argumento (por lo general su puntero) al invocar la primera.

A priori, la opción centralizada resultaba la más conveniente ya que se pretendía preparar el desarrollo para incluir a futuro un sistema de configuración por hardware (botones y pantalla LCD); contar con el conjunto de los mensajes de control MIDI y con su procesamiento en un solo dispositivo facilitaba la consistencia de todo el sistema frente a cambios de configuración en tiempo de ejecución. Sin embargo, esta organización implicaba complejizar el desarrollo (al menos de la unidad MC central), debiendo incursionar en el uso de un nuevo protocolo de comunicación, para una utilidad que no se implementaría de forma inmediata, por lo que optó por la opción descentralizada.

Respecto la distribución de la señal MIDI se testeó la realización por software (opción disponible por defecto en la librería utilizada) pero se descartó inmediatamente ya que repercutía en una pérdida de rendimiento notable en las pruebas de estrés con ambos MC. Finalmente se optó por una estructura tipo divisor MIDI (MIDI Splitter o caja MIDI Thru) realizado por hardware.

En síntesis los MC no cuentan con una instancia de centralización sino que operan como sistemas paralelos completamente independientes uno de otro.

2.2.3.2 Estructura del programa

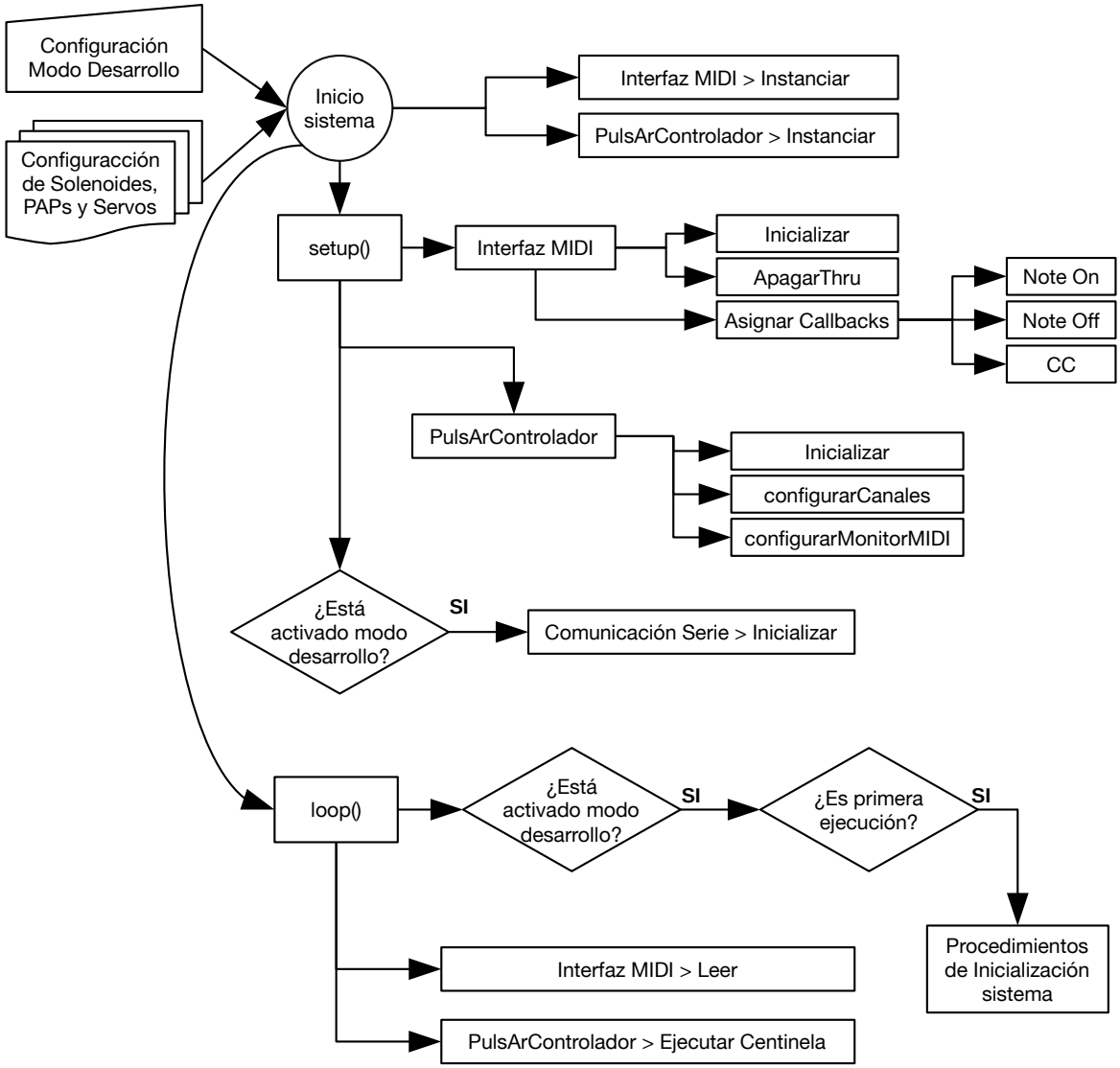

Fig. 14: Esquema de funcionamiento del programa final v3 del MC

2.2.3.3 Clase PulsArControlador

Es la clase que opera como interfaz entre el dispositivo de recepción de mensajes MIDI y las instancias ulteriores de control de motores. Dispone los *callbacks* para la interfaz MIDI frente a recepción de Note On, Note Off y CC. A su vez instancia las clases responsables de generar las señales de control de drivers de motor. Finalmente, se encarga generar un monitor MIDI mediante un LED dispuesto en un pin del MC.

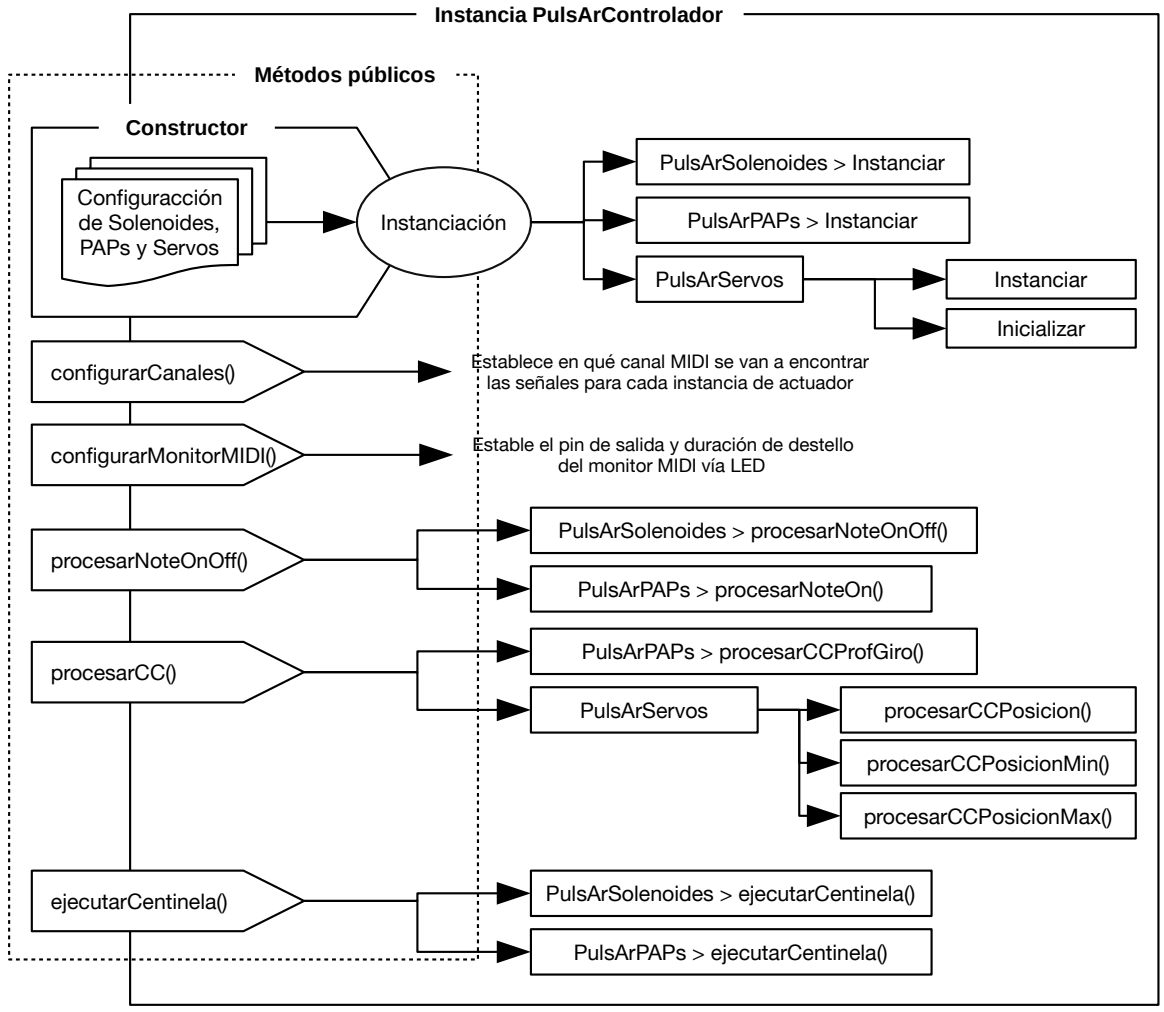

Fig. 15: Esquema Clase PulsArControlador

2.2.3.4 Clase PulsArSolenoide

Es la encargada de controlar los motores solenoides lineales y los motores rotativos de corriente continua (usados en este proyecto de manera equivalente a solenoides rotativos). Configura los pines correspondientes del MC como salidas digitales, genera las señales eléctricas de control de drivers y ejecuta los procedimientos de seguridad establecidos para evitar sobrecarga de fuente, driver y motor.

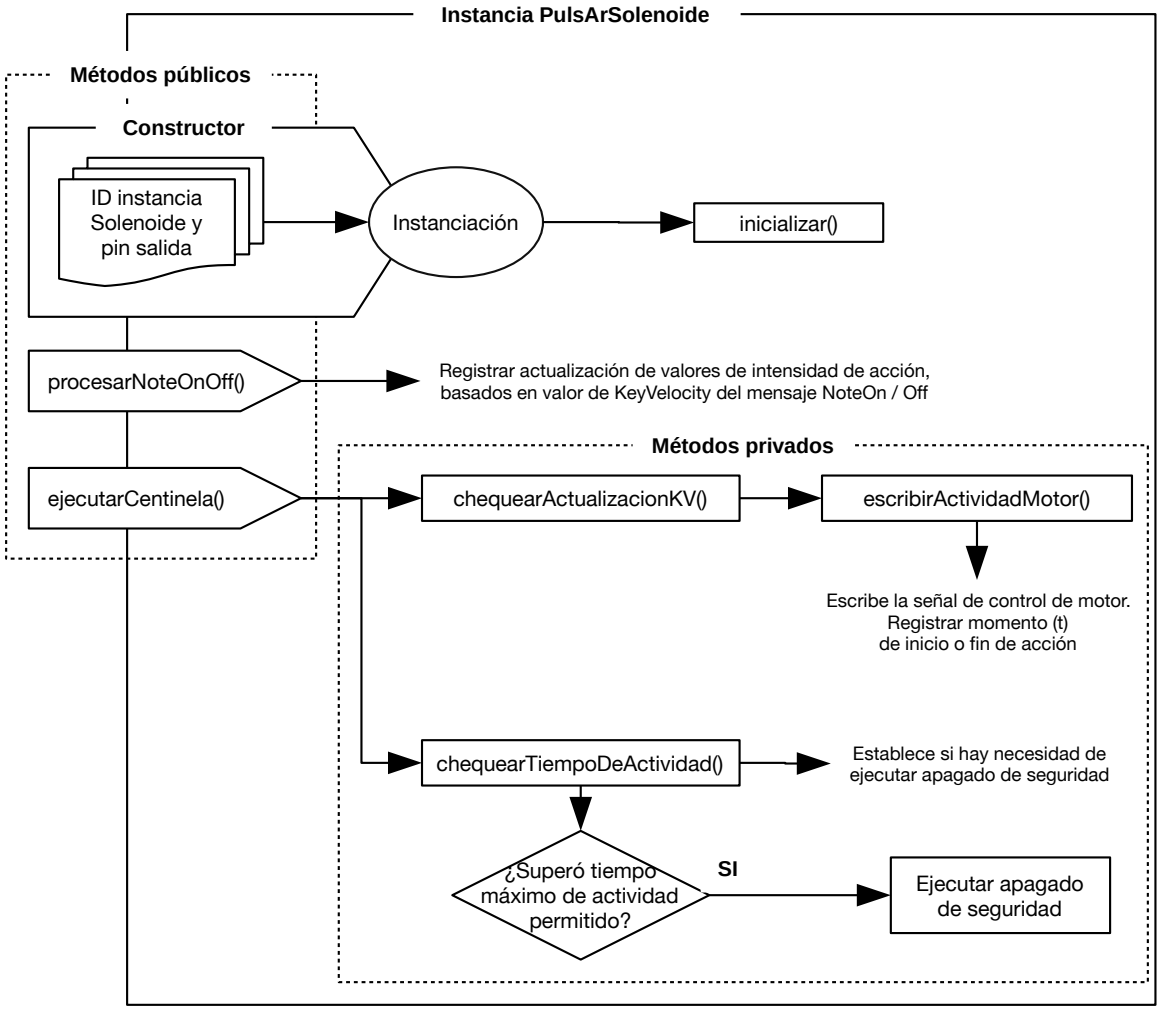

Fig. 16: Esquema clase PulsArSolenoide

2.2.3.5 Clase PulsArPAP

Es la encargada de controlar los motores PAP. Invoca la Librería AccelStepper para generación de las señales eléctricas de control de drivers drv8825 o similar. Contiene los métodos necesarios para traducir los mensajes de control MIDI ingresados en invocación a métodos de la librería citada. Ejecuta procedimientos de control de inactividad y apagado de seguridad.

Cabe señalar que la librería AccelStepper es un software muy completo, tanto por la implementación de algoritmos de aceleración como por incorporar múltiples modos de operación de motores PAP, con diversas configuraciones de señales de salida, disponibles para manejar, por ejemplo, un driver tipo Doble Puente H (L298N o similar) o bien un driver con mecanismo de control de paso y dirección, como el drv8825 utilizado.

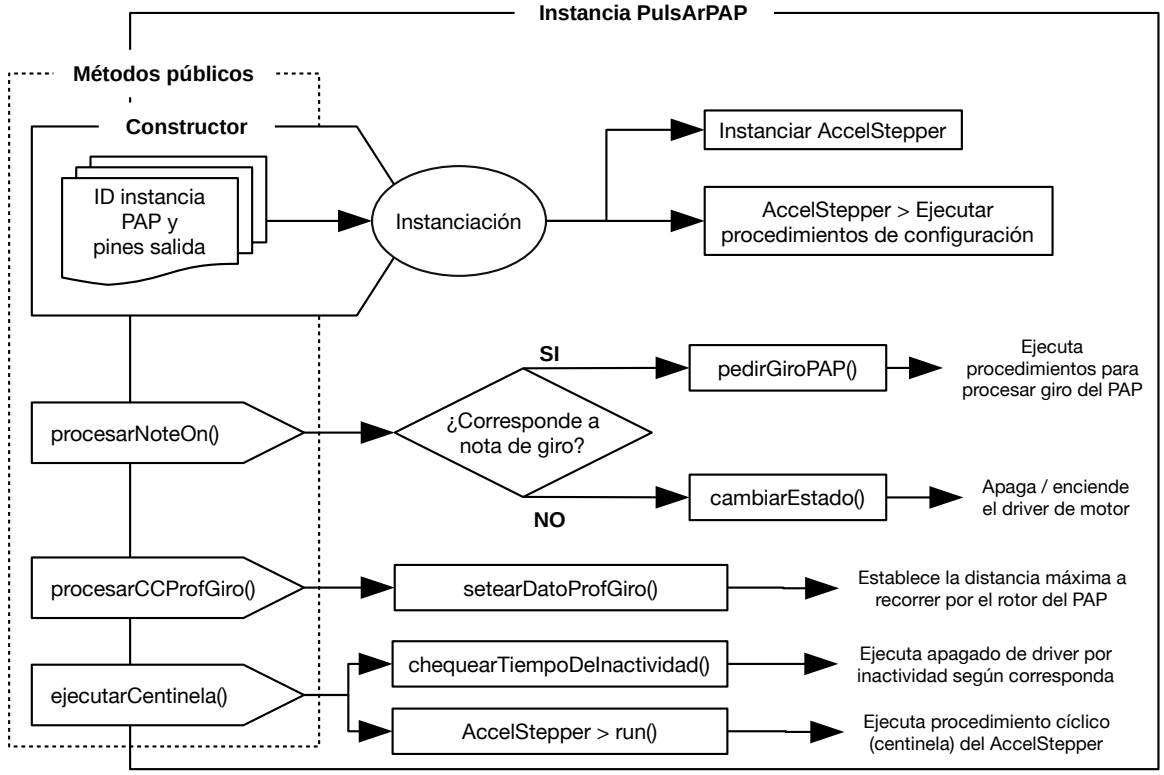

Fig. 17: Esquema clase PulsArPAP

2.2.3.6 Clase PulsArServo

Se encarga de generar las señales de control de los servos Sg90 y Mg996r utilizados en el proyecto. Instancia y configura la librería ESP32Servo. Traduce los mensajes MIDI CC a, por un lado, posiciones angulares del rotor de los motores, y a su vez, a valores de posición máxima y mínima sobre los cuales se hará el mapeo de la posición definitiva del servo.

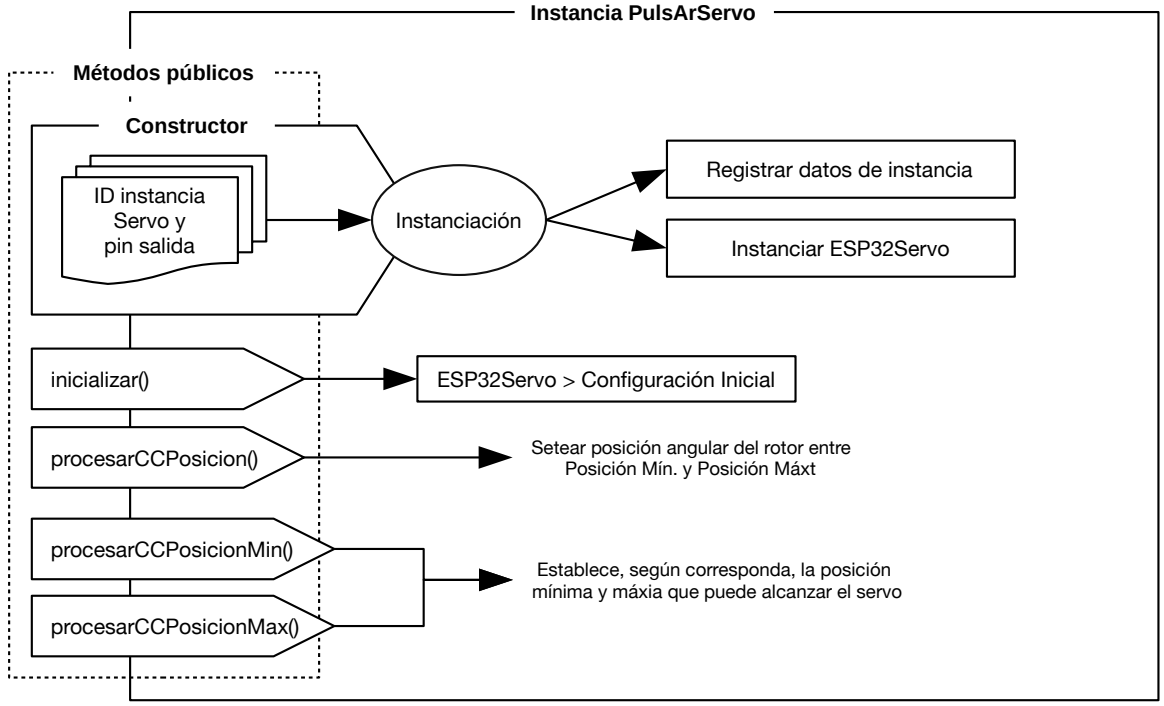

Fig. 18: Esquema clase PulsArServo

2.2.4 Resultados obtenidos en fase definitiva

Las mejoras introducidas luego de la fase prototipo y la depuración lograda mediante v2 redundaron en un sistema satisfactorio, de sencillo mantenimiento y accesible para mejoras futuras. Entre los resultados principales se encuentran:

- Un equilibrio adecuado entre facilidad para el desarrollo y mantenimiento y performance del programa. Acierto en la implementación de paradigma POO, permitiendo un código accesible.
- El avance con el sistema de control de versiones y entorno de desarrollo facilitaron enormemente el procesos de trabajo.
- En particular, la incorporación de AccelStepper, simplificó enormemente la operación de los motores PAP.
- El desdoblamiento de microcontroladores permitió una muy buena performance junto con la posibilidad de controlar más instrumentos y expandir el sistema.

- La estructura del código elegida permitió una rápida configuración del funcionamiento de cada microcontrolador.

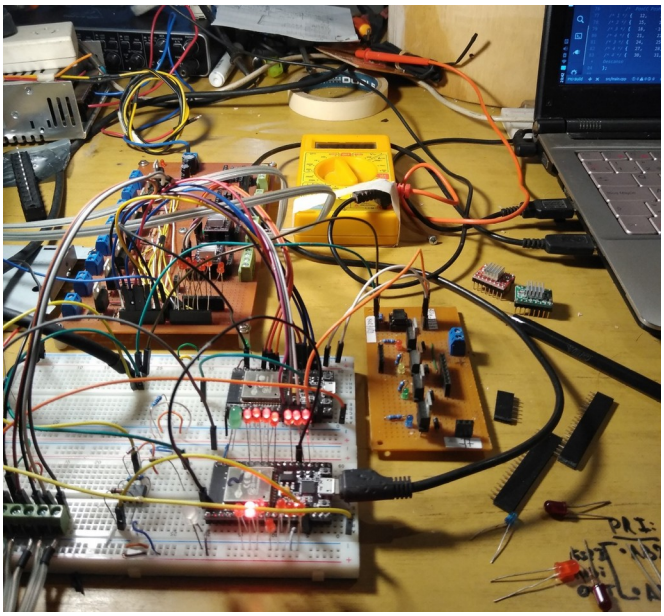

Fig. 19: Pruebas de estrés de MCs

A futuro, se plantean varias mejoras por incluir, como ser la posibilidad de contar con un mecanismo de configuración de parámetros en tiempo de ejecución, vía interfaz de control por hardware (mediante botones y LCD en el gabinete) o bien vía web, aprovechando las opciones de conectividad que provee el ESP32. Sin embargo, el programa para la operación de MCs alcanzado en versión definitiva cumplió de forma adecuada con las expectativas planteadas al inicio del desarrollo.

3. Electrónica

Entre los circuitos electrónicos desarrollados para el proyecto se encuentran un dispositivo MIDI Splitter, las entradas MIDI al MC, las placas para los diversos drivers de motor utilizados en el dispositivo, un sencillo sistema de monitoreo mediante indicadores LED tanto de la operación de motores como del ingreso de datos MIDI y el interconexionado global entre las partes. Cabe mencionar que para las fuentes de alimentación eléctrica se utilizaron diversos dispositivos cerrados, tanto recuperados como adquiridos especialmente para el proyecto. Al igual que la programación y los dispositivos mecánicos, esta dimensión del proyecto paso por fase de prototipado y desarrollo final, las cuales se detallan a continuación.

3.1. Fase prototipo

3.1.1 Drivers para motor solenoide

Los drivers para motor solenoide lineal o rotativo pudieron ser rápidamente resueltos de manera definitiva. Se trabajó con un modelo de circuito planteado por G. W. Raes[41], el cual también aparece en trabajos de A. Kapur. En el diseño definitivo se optó por utilizar la versión sin mecanismo de retención. En la construcción, se usó el transistor tipo Darlington TIP122, por disponibilidad dentro de los componentes recuperados y por sus buenas prestaciones (ej.: soporta una corriente de hasta 5 amperes).

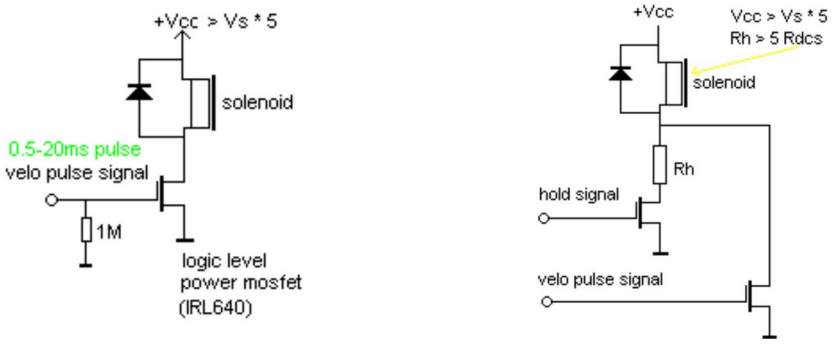

Fig. 20: Circuito planteado por G.W. Raes como driver para solenoide lineal con y sin mecanismo de retención[42].

Luego de pruebas exitosas con *protoboard* se construyó un módulo individual en placa de desarrollo para evaluar la arquitectura definitiva. Se incluyó una bornera para salida, una resistencia de 220Ω en un esquema de bifurcación de su entrada para disponer un LED como monitor de operación, y un diodo 1N4148 como mecanismo de protección del puerto del microcontrolador.

41 En "Expression control in automated musical instruments. An ongoing survey and report". Dr. Godfried-Willem Raes. Última revisión, Marzo de 2020.

42 El circuito con sistema de retención planteado por el autor prevé el uso de 2 líneas de acción sobre el actuador, una

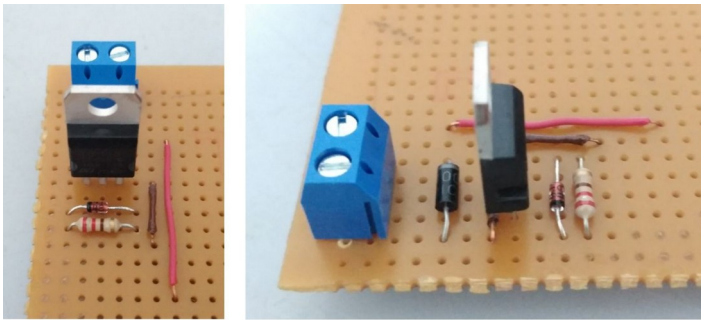

Fig. 21: Driver solenoide, pruebas en placa de desarrollo

Siendo también satisfactorias las pruebas con este driver se avanzó a la construcción de la placa de testeo definitiva para drivers solenoide, la cual también contendría la entrada MIDI para el MC (ver más abajo).

3.1.2 Entrada MIDI

El circuito para ingresar la señal MIDI al MC se construyó basándose en el circuito sugerido por la Especificación MIDI 1.0[43] de 1996.

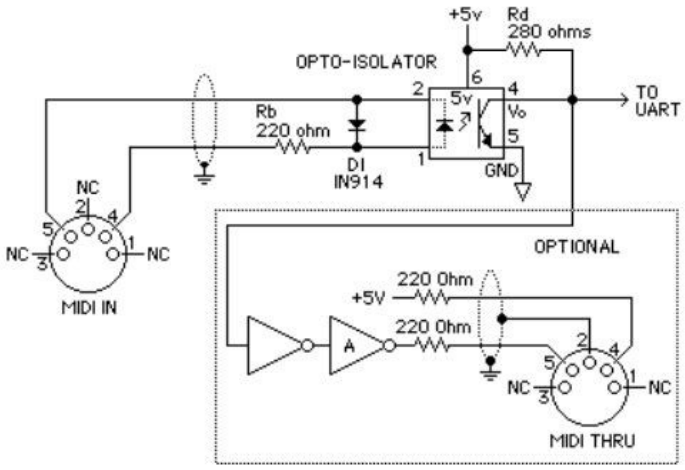

Fig. 22: Implementación de puertos físicos MIDI según Especificaciones MIDI 1.0 de 1996

Además de quitar el módulo de MIDI Thru, se reemplazó el opto-aislador sugerido 6N138 por uno más rápido y pin-compatible 6N139, por falta de stock del integrado requerido.

43 "*The Complete MIDI 1.0 Detailed Specification. Incorporating all Recommended Practices document version 96.1, third edition*". Disponible en www.midi.org

3.1.3 Placa de testeo definitiva MIDI + Drivers Solenoide

Finalmente, para facilitar el proceso de transporte, conexionado y testeos, se construyó una placa de desarrollo consistente en 4 drivers solenoides según el modelo de circuito antes citado y un puerto de Entrada MIDI también basado en el circuito descrito anteriormente.

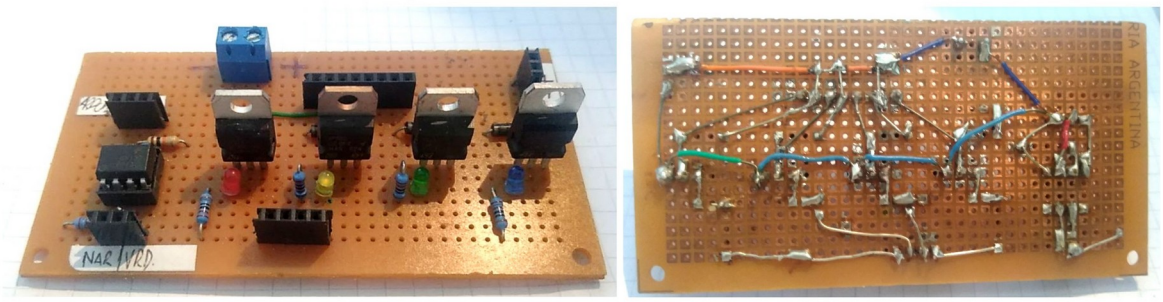

Fig. 23: Placa Drivers Solenoides + Entrada MIDI

3.1.4 Drivers preliminares para motor PAP

El trabajo con drivers para motor PAP evidenció un proceso ligado, más que al desarrollo propio de un circuito superador, al aprendizaje respecto del funcionamiento de los modelos de circuito típicos utilizados para el control de este tipo de motores.

3.1.4.1 Puente H simple y doble construido con componentes discretos

Se comenzó trabajando con modelo típico de circuito llamado *Puente H*, el cual sirve para alimentar un motor de corriente continua (bobina simple) pudiendo controlar su dirección de giro con el encendido y apagado de una tensión de control en un sector del circuito (algo factible de realizarse desde el MC), sin la necesidad alterar la estructura física del mismo.

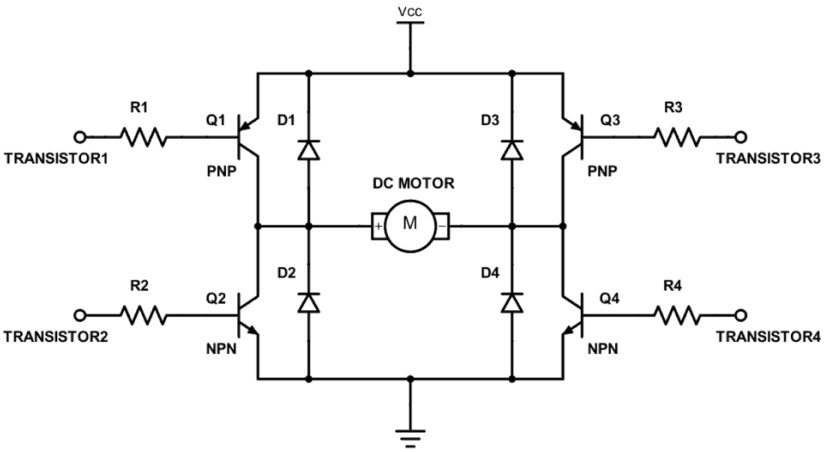

Fig. 24: Circuito básico Puente H para acción con control de dirección de Motor CC

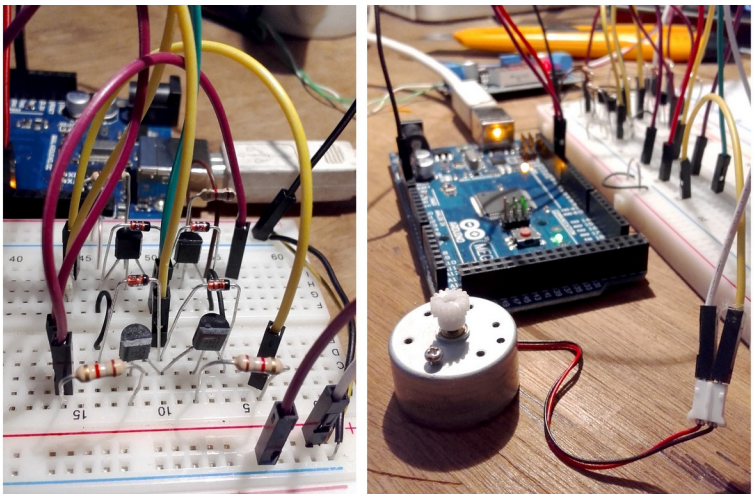

Fig. 25: Armado y testeo de circuito Puente H en motor CC

Luego de pruebas en protoboard con *Puente H Simple* (controlando motores de CC) construido con componentes discretos, se avanzó en testeos de control de motores PAP con un circuito del tipo *Doble Puente H* usado para motores PAP Bipolares[44].

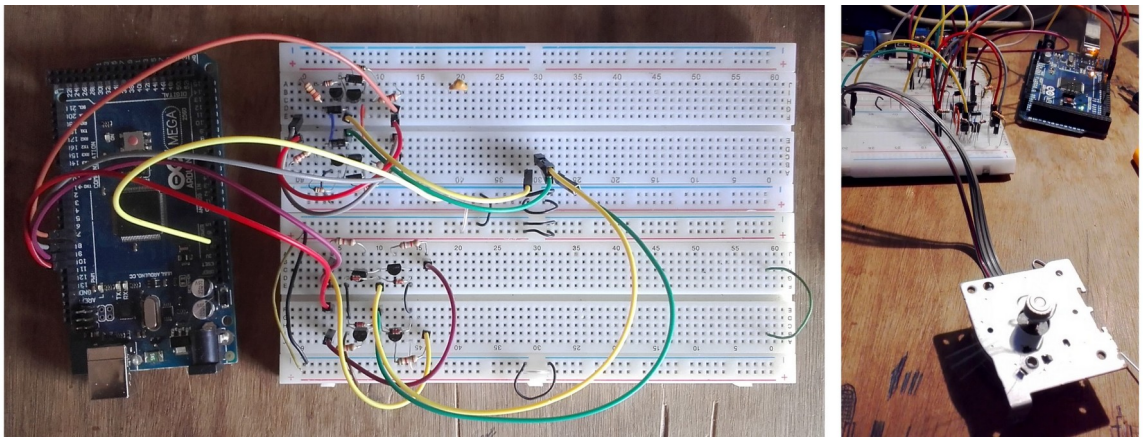

Fig. 26: Armado y testeo de circuito Doble Puente H en motor PAP Bipolar

La utilización de estos circuitos resultó de alto valor pedagógico ya que permitió comprender la construcción y funcionamiento de lo que es el corazón del tipo de driver utilizado en la fase definitiva. Sin embargo, en este punto se observó que la complejidad de construcción, testeo, cálculo del rendimiento / consumo de corriente eléctrica, etc. volvía a esta estrategia poco viable para su utilización definitiva en el proyecto.

44 No es menester de este trabajo explicar las características constructivas, de funcionamiento y control de motores PAP Bipolares y Unipolares. Solo diremos, que los primero son más sencillos de operar y por tanto su driver es más sencillo de construir.

3.1.4.2 Trabajo con driver basado en L293B y L298N

Atendiendo a la complejidad antes mencionada, se procedió a realizar pruebas con circuitos integrados que resuelvan de manera más directa y sencilla el control de motores bipolares. Se trabajó con 2 integrados: L293B y L298N, ambos con una estructura de *Doble Puente H*.

El primero de estos proveía una corriente relativamente baja (1 amper) para las pruebas en curso con motores PAP recuperados que no poseían especificaciones técnicas, por lo que para contar con mayor margen de trabajo se pasó a utilizar el L298N con corriente de salida máxima de 4 amperes.

Se utilizó un integrado tipo Multiwatt15 de 15 pines. Para el mismo se construyó una placa de desarrollo incorporando el puñado de componentes discretos extra que requiere.

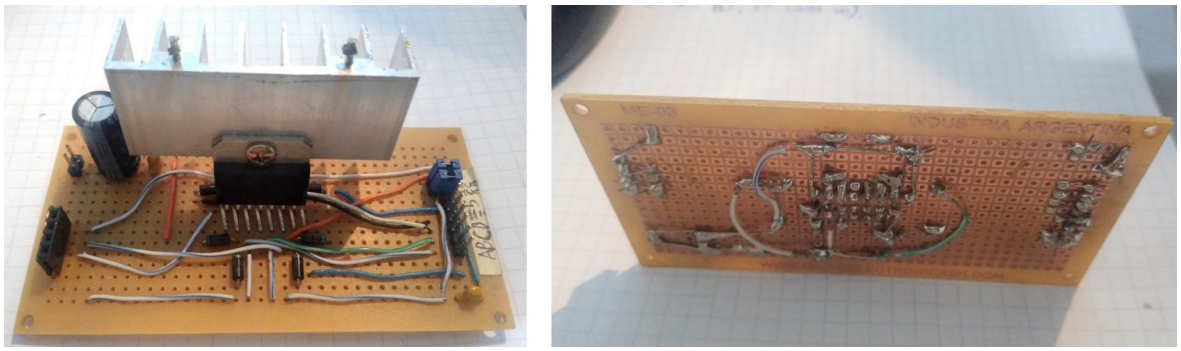

Fig. 27: Driver PAP basado en integrado L298N terminado, con disipador incorporado

Si bien este driver resuelve una buena parte de los problemas relativos a construcción, implica el uso de varios componentes discretos adicionales, ocupa un tamaño considerable y su precio es elevado. A su vez, las pruebas realizadas con motores PAP no propiciaban una performance óptima que pudiera traducirse en resultados sonoro/musicales adecuados. Sobre esta base se avanzó a lo que sería la solución definitiva en cuanto a drivers PAP.

3.1.5 Trabajo con drivers símil Pololu a4988 y drv8825

A partir de la difusión de iniciativas como el proyecto RepRap[45] orientadas a la construcción de impresoras 3D, routers CNC y dispositivos similares, con un enfoque de soft y hardware abierto, se masificó el uso de drivers para motor paso para motores tipo NEMA de alto rendimiento y con posibilidad de uso de la técnica de micro-paso, en una solución "todo en uno" consistente en una pequeña placa de relativamente bajo costo, fácil utilización y de un tamaño en el orden de 1 pulgada cúbica. Sin dudas, la estrategia definitiva de manejo de motores PAP incluiría esta tecnología.

Se hicieron testeos con 2 tipos de driver: símil Pololu a4988 y símil Pololu drv8825. Ambos pin-compatibles.

45 https://reprap.org/wiki/RepRap/es

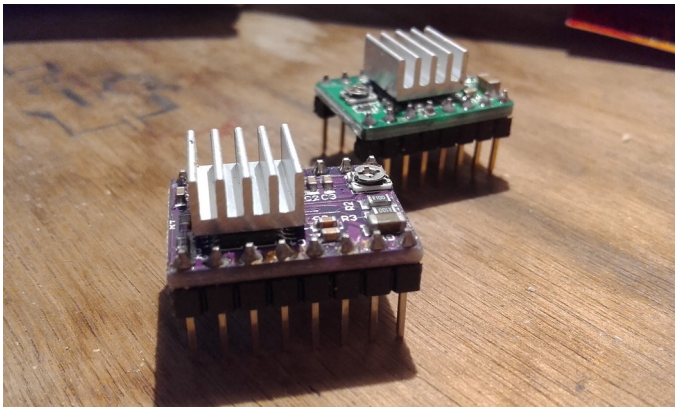

Fig. 28: Drivers PAP símil Pololu a4988 (verde) y símil Pololu drv8825 (violeta), con sus respectivos disipadores de calor.

La experiencia con el a4988 fue negativa. Cabe mencionar que la operación con ambos implica un calibrado del limitador de corriente incorporado mediante un potenciómetro SMD presente en la placa. Sin embargo, el calibrado a partir de las especificaciones técnicas conseguidas obtuvo resultados dispares, quemándose varias placas. Además la configuración de micro-paso en este driver es más limitada. A partir de esto, se pasó a realizar pruebas con el drv8825.

El drv8825 posee una corriente de trabajo máxima superior a su hermano, alcanzando los 2.2 amperes con disipador, permite trabajar en resoluciones de hasta 1/32 de paso y alcanza los 45 voltios de salida. A su vez, el proceso de calibrado es bastante más sencillo por lo que pasó a ser la opción definitiva.

Para la fase de prototipado se trabajó con estos drivers siempre con *protoboard*, quedando para la fase definitiva la construcción de una placa que los contenga.

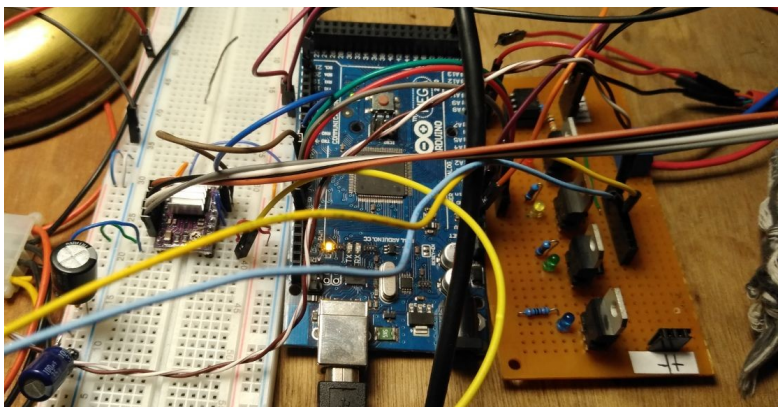

Fig. 29: Drivers PAP símil Pololu a4988 (verde) y símil Pololu drv8825 (violeta), con sus respectivos disipadores de calor.

Luego de pruebas de rendimiento se concluyó que, para el motor PM55L utilizado en el proyecto, la configuración de ⅛ de paso es la que permite un equilibrio adecuado entre

movimiento fluido, con la menor cantidad de vibraciones y ruido por artefactos, buena capacidad de retención, etc.

Es importante decir que el proceso de calibrado del motor[46] en cuanto a limitación de corriente máxima es crucial para la protección del motor y del driver, evitando sobrecarga (evidenciada en recalentamiento excesivo del motor). Se logró conseguir la hoja de datos del motor recuperado utilizado, por lo que se tenía el valor exacto de su máxima corriente soportada, 800mA, y se logró un ajuste preciso.

3.1.6 Resultados obtenidos en fase prototipo

- El uso de drivers símil Pololu drv8825 es una solución incuestionable, la cual provee enorme facilidad para su adquisición, integración al resto del proyecto, operación, etc. frente a las demás estrategias utilizadas con componentes discretos u otros integrados. Suma a su vez la posibilidad de implementar la técnica de micro-paso, de manera sencilla sin necesitar otros componentes.
- El circuito para los drivers de motor solenoide resultó muy efectivo desde un primer momento y sería replicado sin mayores modificaciones en la versión definitiva. Cabe destacar que hasta este punto se utilizaba PWM como mecanismo de control de intensidad del golpe, lo cual será descartado a futuro. Quedarán pendientes para futuros desarrollos probar la solución completa planteada por Raes para circuitos con retención.
- El reemplazo eventual del 6N138 por el 6N139 en circuito de recepción MIDI no mostró inconvenientes.
- A fin de testear de forma extensiva el desarrollo, de realizar presentaciones públicas de los avances del mismo, etc. fue crucial el superar en algunas partes del circuito el estadio de "*en protoboard*". Esto se relaciona con la conclusión de contar con prototipos cerrados y lo más plenamente funcionales para el abordaje cabal del objeto de estudio. Se tematiza más profundamente este aspecto de orden metodológico al final del documento.

3.2. Fase definitiva

La fase definitiva del desarrollo de la electrónica del proyecto implico un salto importante desde la fase prototipo. Esta consistió en la fabricación de:

- Una placa "PulsAr Madre" que alojaría los 2 MCs junto con sus respectivos circuitos de entrada MIDI, sistema de alimentación, salida de monitoreo LED, etc.
- Una placa de drivers de motor, para solenoides y PAPs, según se describió en fase prototipo.
- Una segunda placa, integrando drivers de motor para solenoides, junto con una "terminal de embarque" para la operaciones de motores Servo.
- 3 placas de soporte de monitoreo LED
- 2 placas conexionado de Servos para tambores
- Gabinete metálico para alojamiento de placas MCs, drivers y monitoreo, junto con fuentes de alimentación para drivers y MCs (+24V y +5V), puertos de conexionado

46 Por su enorme claridad didáctica se recomienda el video explicativo de Pololu para realizar la calibración del motor: https://www.youtube.com/watch?v=89BHS9hfSUk

hacia actuadores, sistema de interconexión de placas desarrolladas y alimentación eléctrica doméstica (220V).

Cabe mencionar, como rasgo característico de todas las placas desarrolladas, que las mismas se diseñaron teniendo en cuenta la accesibilidad para los procesos de corrección de errores, y la evolución constante. En muchos casos se incluyen dobles puertos de salida para testeos, siendo el caso más paradigmático el de la estructura hackeable de la placa madre.

3.2.1 Placa PulsAr-Madre

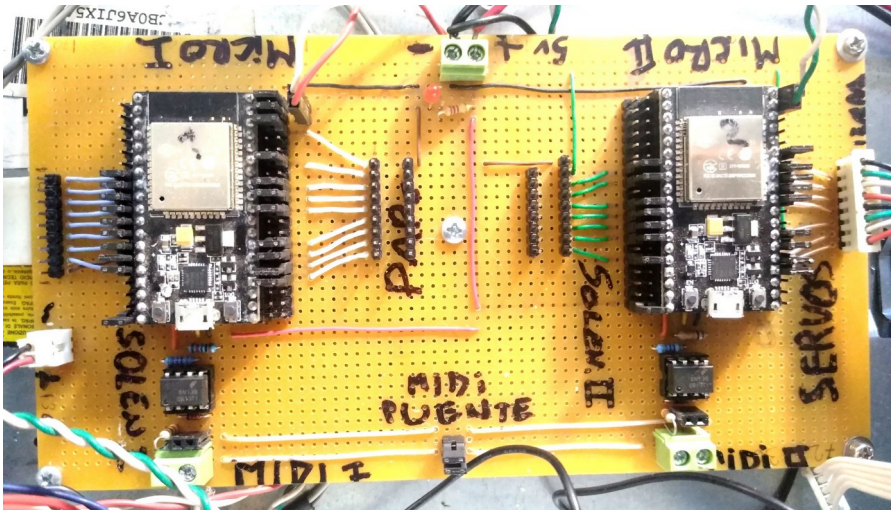

Fig. 30: Placa PulsAr Madre

Fue construida en una placa de desarrollo perforada de 72 x 36 orificios. Es el centro de control del sistema creado, ya que aloja las unidades MC. Posee las siguientes características:

- Zócalo para 2 MC ESP32
- Entrada de alimentación de 5V por bornes, con LED indicador energía
- 2 Puerto Entrada MIDI basados en el opto-aislador 6N139, con conexión por bornes o pines hembra.
- Sistema de Puente MIDI, con activador vía jumpers
- Salidas duplicadas hacia actuadores mediante pines macho.
- Conexión de tierra y tensión de referencia del MC 1 hacia la placa de drivers I
- Sistema de conexión de MCs hackeable vía jumpers (se detalla más abajo)

Los puertos de Entrada MIDI fueron diseñados siguiendo el mismo circuito utilizado en la fase prototipo. Dado que al momento de su construcción no se pudo conseguir el integrado 74HC14N[47] requerido para hacer el Divisor MIDI (Splitter) adecuado para una estructura tipo

47 Un inversor Schmitt trigger sextuple de respuesta rápida.

Thru[48], se incluyo un sistema rudimentario de puente MIDI directo, activable mediante 2 jumpers, el cual envía la misma señal de entrada MIDI a sendos puertos de los MCs.

Los zócalos para MCs fueron construidos mediante 38 pines hembra cada uno. El conexionado hacia los puertos de salida se hizo siguiendo la mejor disposición encontrada de acuerdo a los pines disponibles junto con una configuración funcional entre sí[49]. A su vez, dichas salidas están duplicadas a fin de proveer puertos de conexión tanto hacia drivers / motores como hacia los monitores LED del panel frontal del gabinete.

Cabe mencionar que para alimentar dichas placas de monitoreo, se evidenció innecesario incluir las correspondientes resistencias en serie para limitar el flujo de corriente y proteger los LEDs. A su vez, no se registraron problemas de funcionamiento al alimentar de manera directa desde un mismo pin del MC, tanto los LEDs como los drivers de motor. Sin embargo, este formato no se utilizó de cara a los servomotores (es decir, no tienen monitoreo en el panel frontal del gabinete), ya que estos se conectan de forma directa, desde el MC hasta el ingreso de datos del motor, pasando por un cable de 3 mts. de longitud, previniendo así eventuales pérdidas en la señal si, a su vez, se incluía un LED sin un driver adicional.

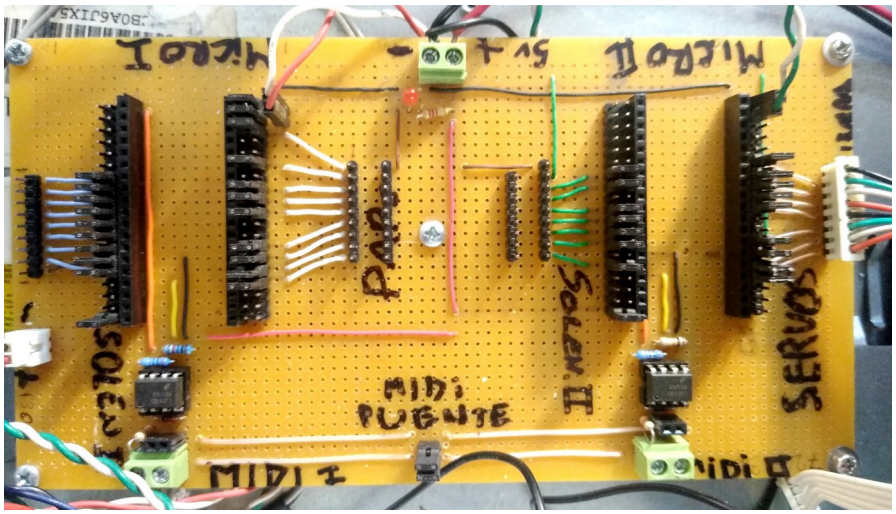

Fig. 31: Placa PulsAr Madre, vista sin los MCs.

3.2.1.1 Arquitectura hackeable

Una de las características destacadas de esta placa es la de contar con una arquitectura hackeable *on-board*. Es decir que la placa, en su construcción inherente, permite ser intervenida a futuro. Cada pin hembra de los respectivos MCs se conecta directamente a un pin macho el cual esta dispuesto a su vez junto a otro, que finalmente se conecta al resto del circuito. El cierre del circuito entonces se realiza mediante jumpers individuales por cada pin. Esto permite que, si

48 Cabe mencionar que se hicieron pruebas con otro inversor Schmitt trigger sextuple, el 40106, con resultados negativos, dados los tiempos de respuesta insuficientes de dicho integrado.

49 No todos los pines digitales del ESP32 pueden ofrecer el conjunto de sus funcionalidades como puerto digital individual, al utilizarse simultáneamente. A su vez, se han evidenciado errores de funcionamiento en pines específicos al destinarse a control de servomotores.

bien existe una disposición de conexiones fija que supera un "modo protoboard" evitando posibilidad de desconexiones, cortos, etc., en una eventual necesidad de cambio de la asignación de pines del MC, un fallo específico, la inclusión de un nuevo circuito, etc., simplemente mediante la desconexión del jumper correspondiente se puede redireccionar dicho pin a otro circuito y por tanto a otra funcionalidad.

Esta funcionalidad se evidenció adecuada para un desarrollo de naturaleza exploratoria, en permanente evolución, sujeto a experimentaciones, pero a su vez robusto, transportable, como requiere este proyecto.

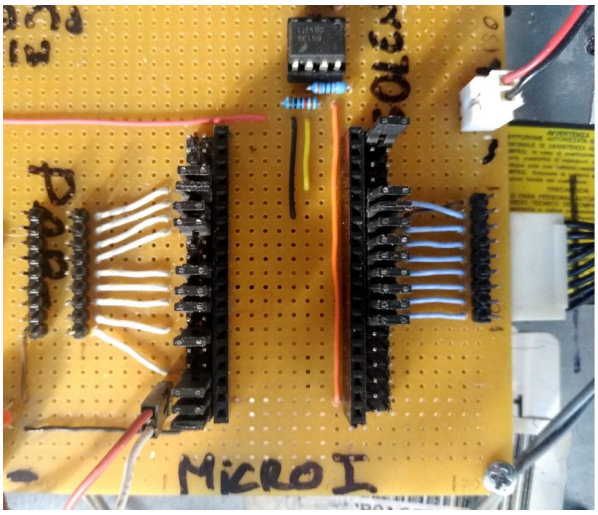

Fig. 32: Detalle arquitectura hackeable vía jumpers y salidas duplicadas en Placa PulsAr-Madre

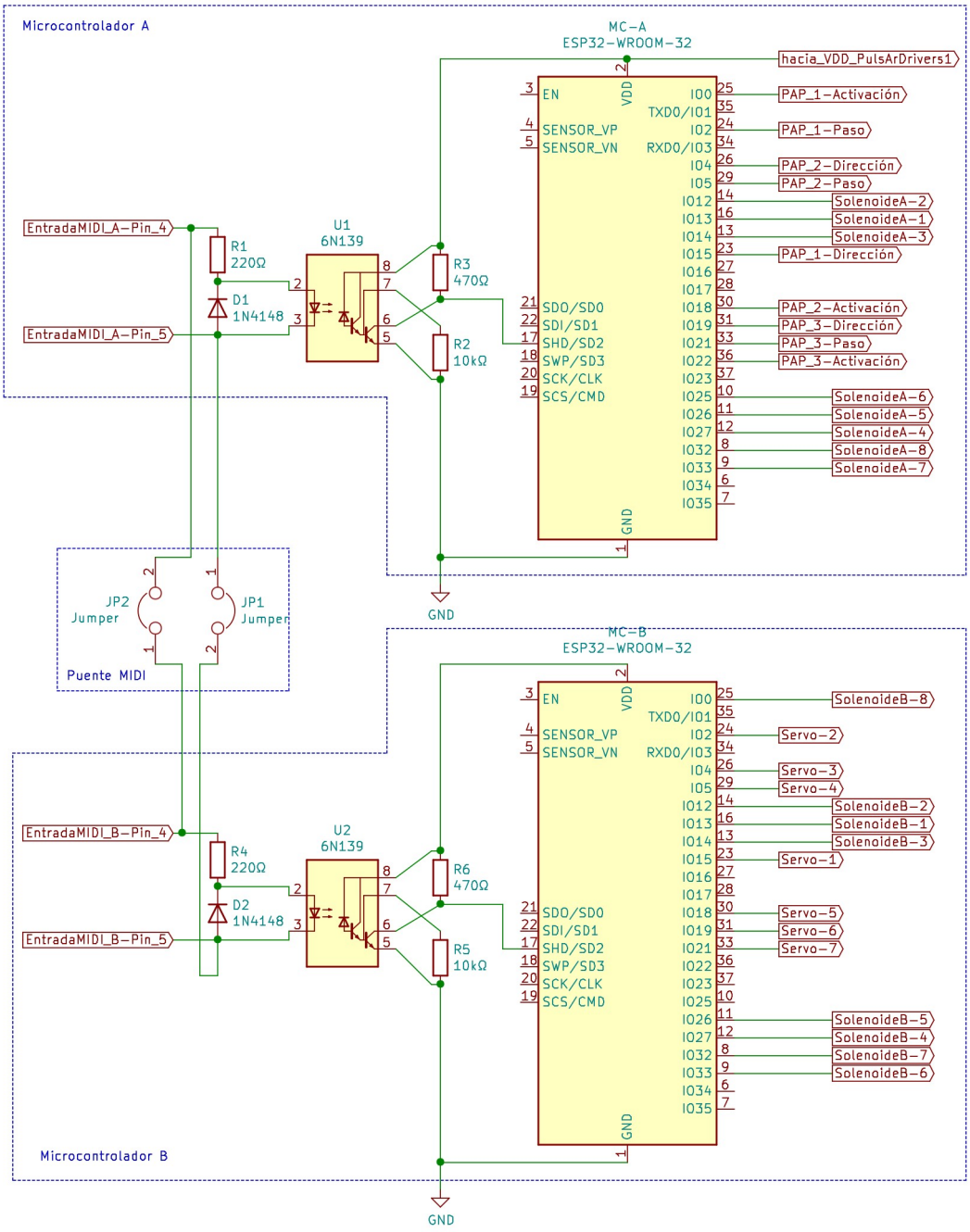

Fig. 33: Esquemático circuito placa PulsAr-Madre.

3.2.2 Placa PulsAr-Drivers I

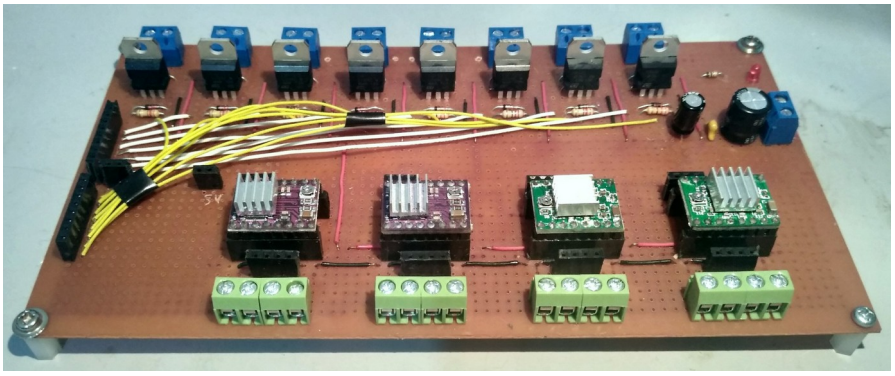

Fig. 34: PulsAr-Drivers I, vista superior

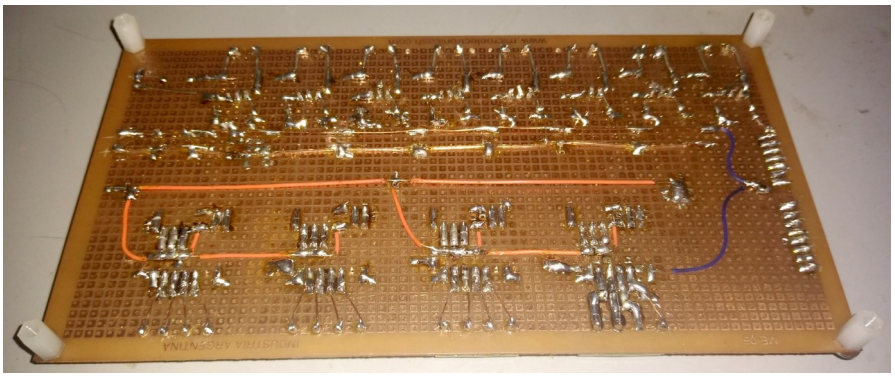

Fig. 35: PulsAr-Drivers I, vista inferior

Al igual que la placa madre, PulsAr-Drivers I fue construida en una placa de desarrollo perforada de 72 x 36 orificios. Posee:

- Entradas de señales de control desde el MC mediante pines macho
- Entrada de alimentación eléctrica de 24V para motores, con protección de picos de tensión mediante capacitores en paralelo.
- 8 drivers de solenoide como los descritos anteriormente basados en Transistor Darlington TIP122, con salidas por borne
- Salidas de led de monitoreo para solenoides, finalmente no utilizadas.
- 4 zócalos para drivers PAP tipo drv8825 o similar pin-compatible, con entradas mediante pin hembra (paso, dirección, activación) y salidas mediante borne y pin hembra

A fin de poder modificar la configuración independiente de micro-paso por cada driver, se implemento un sistema de configuración mediante jumpers, por debajo de la placa de cada driver.

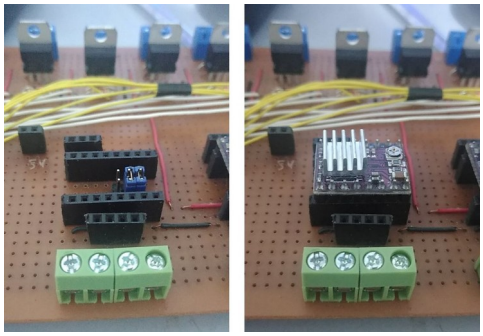

Fig. 36: PulsAr-Drivers I, Detalle configuración micro-paso mediante jumpers

3.2.3 Placa PulsAr-Drivers II

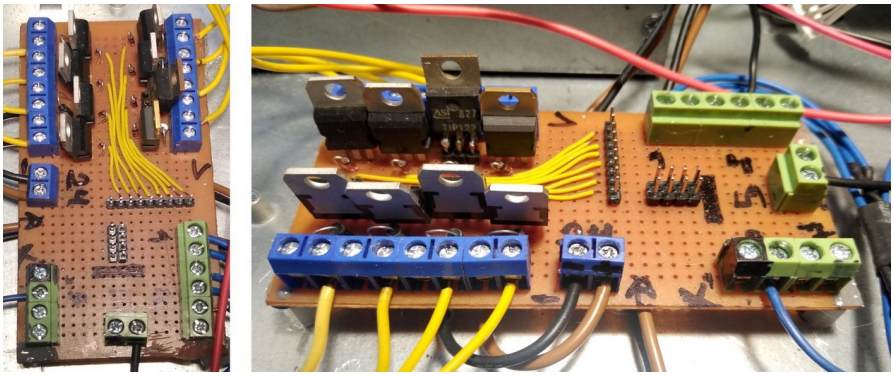

Fig. 37: Placa PulsAr-Drivers II, vista superior y lateral

En lo que a control de motores respecta, se construyó una segunda placa pensada como "puerto de embarque" de cara a los servomotores[50] y en la que se agregaron drivers de solenoide, manteniendo el mismo diseño que en el resto del proyecto. En este caso, se utilizó una placa de desarrollo perforada de 16x36 orificios, lo cual implicó un gran desafío de construcción y sobre todo de soldadura, al tener que incluir en un espacio muy reducido:

- Entrada de señales de control desde el MC mediante pines macho
- Entrada diferenciada de 5V y 24V, para alimentación de motores, con tierra independiente, mediante bornes.
- Salidas para 7 servomotores, de las cuales solo se utilizarán 5 en las conexiones del gabinete, mediante bornes
- 8 drivers de motor solenoide basados en transistor Darlington TIP122 con salida mediante bornes

50 Hablamos de puerto de embarque ya que de lo que se ocupa esta placa es de preparar las señales para la salida. Esto es: proveer de las terminales tipo borne para conectar a los cables de mayor grosor que van hacia las terminales de salida del gabinete, las cuales hubieran redundado en una asimetría excesiva en el diseño general de la placa madre de haberlas colocado directamente allí, disponer de la tierra común entre la fuente de alimentación de 5V hacia los motores y el MC, configuración obligada para el funcionamiento de los servos.

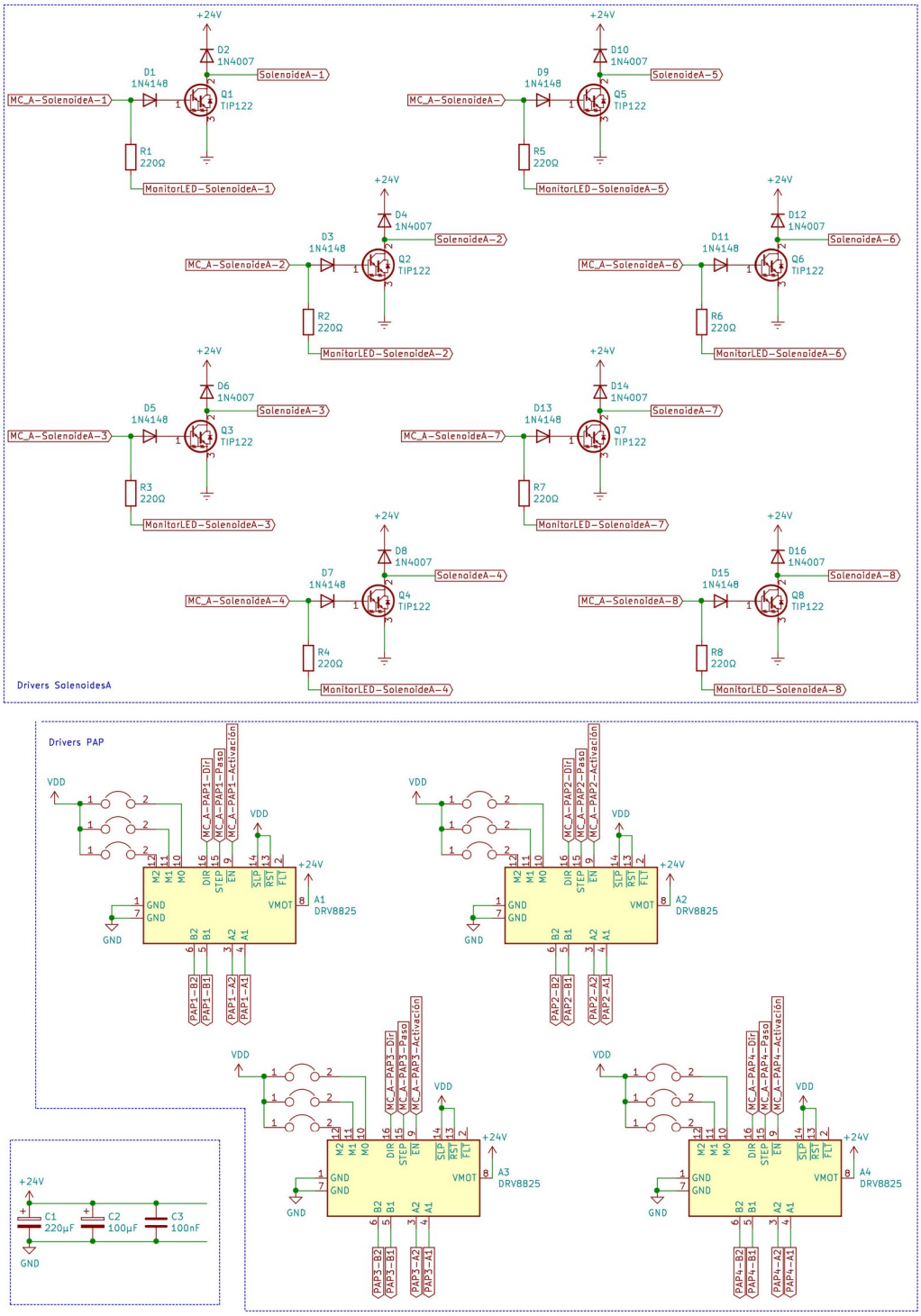

Fig. 38: Esquemático circuito placa PulsAr-Drivers1.

3.2.4 Placa PulsAr-MonitorLED

A fin de ofrecer un monitoreo de LED de la actividad de los actuadores principales (Solenoides y PAPs) se construyeron 3 placas de soporte de LEDs con circuito de cátodo común, usando recortes de placa perforada de otros proyectos.

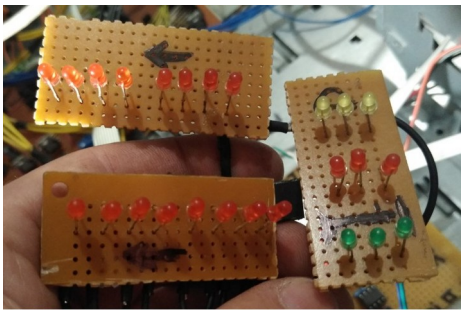

Fig. 39: Placa PulsAr-MonitorLED

3.2.5 Placas PulsAr-RecepServos

Finalmente, se construyeron 2 pequeñas placas de recepción centralizada de las conexiones de los servos, ubicadas directamente sobre el marco soporte de los instrumentos.

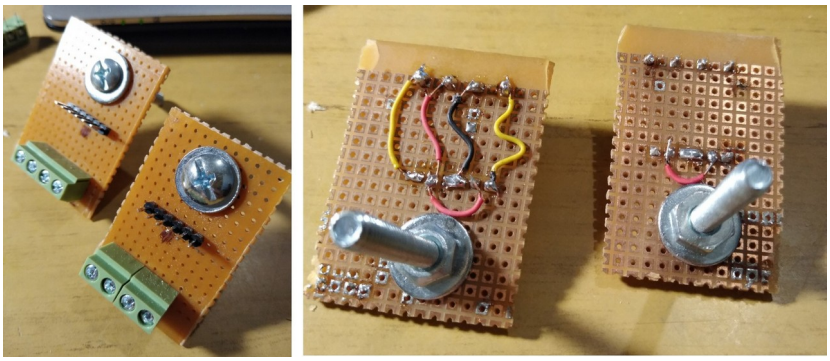

Fig. 40: Vista superior e inferior (durante el proceso de construcción) de Placas PulsAr-RecepServos.

3.2.6 Fuentes de alimentación

Para la alimentación eléctrica del sistema se utilizaron 2 fuentes de alimentación diferenciadas, de +24V y +5V.

Para resolver la alimentación de +24V, se resolvió adquirir una fuente específica capaz de entregar hasta 20 A. Para la de +5V, se utilizó la salida de dicho valor de tensión de una fuente ATX recuperada. A dicho artefacto, además de retirarle la gran cantidad de cables que dispone, se le agrego una linea de salida de 220V desde un interruptor general que disponía el aparato, en serie a la alimentación externa desde la ficha C14, a fin de proveer de tensión a la fuente de 24V, usando un único interruptor que quedaría en la parte trasera del gabinete.

Se decidió adquirir una fuente específica de alto amperaje para alimentar los motores principales del sistema, de +24V, para eliminar toda una serie de posibles problemas de rendimiento que pudieran surgir durante el desarrollo, dado la gran cantidad de márgenes de error y comportamientos no lineales o poco previsibles al ser componentes recuperados la materia prima central de trabajo. Se evidencia que hubiera sido deseable hacer testeos en cuanto a rendimiento con una sola fuente de 24V de mayor amperaje o 2 con idéntico voltaje y amperaje total distribuido, para evitar sobrecargas y obtener mejor performance.

Para obtener los mencionados +24V se indagaron posibles soluciones mediante conexionado en serie de la salida de +12V de 2 fuentes ATX, pero no se realizaron pruebas concretas, dada la poca información que se disponía al respecto.

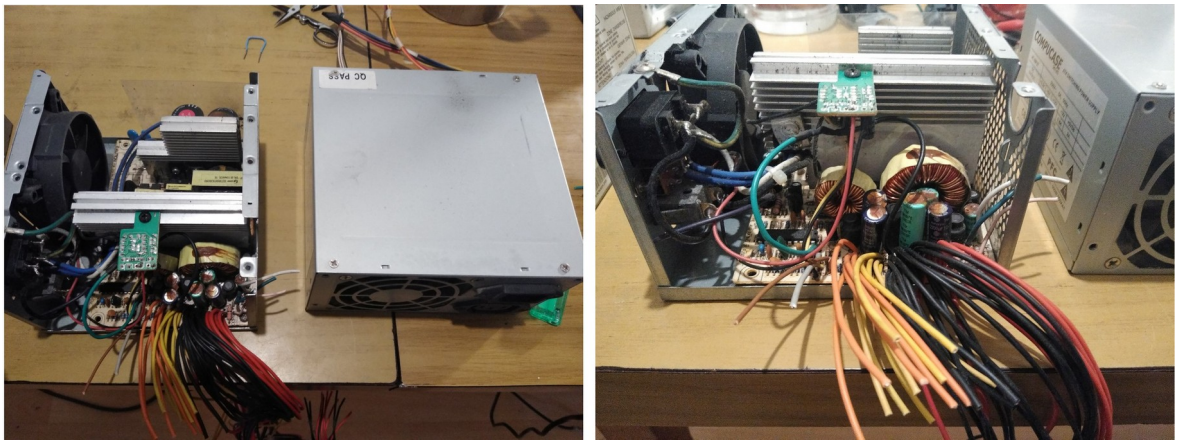

Fig. 41: Preparación de Fuente de alimentación de 5V. En cada foto, a la izquierda se visualiza la fuente ATX utilizada en la versión final, y a la derecha, la utilizada durante la fase de prototipo.

3.2.7 Gabinete PulsAr-CC

Para concentrar el conjunto de la electrónica desarrollada se dispuso de un gabinete metálico construido sobre un gabinete de computadora recuperado. Allí se alojaron las fuentes de alimentación, la entrada MIDI, las diversas placas de MCs y drivers, el sistema de monitoreo por LED y las salidas hacia los actuadores. Toma el nombre de PulsAr-CC, por la sigla de centro de control.

Fig. 42: Gabinete PulsAr-CC

De entre varias opciones del lote de artefactos recuperados, se optó por un viejo gabinete de computadora Compaq, el cual presentaba una enorme rigidez constructiva, algo más de espacio que los gabinetes convencionales y un sistema de apertura y cierre sencillo mediante tornillos con rosca plástica para mano, fijados a la tapa del gabinete. Desafortunadamente la fuente de alimentación que traía el gabinete, también algo más grande que las típicas fuentes ATX y a la medida de su chasis, no funcionaba, por lo que tuvo que ser reemplazada por el dispositivo antes detallado.

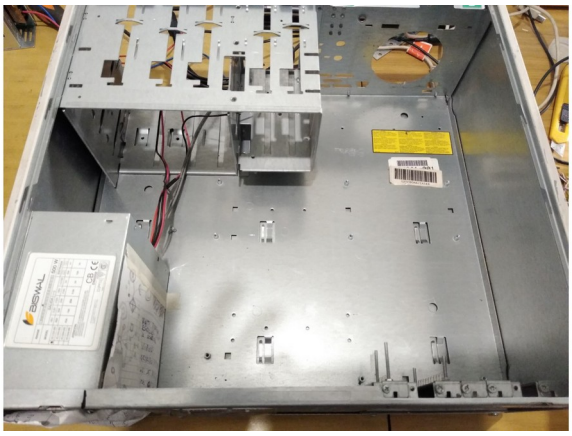

Fig. 43: Gabinete vacío, luego de su vaciado y limpieza y previo a la incorporación de la electrónica.

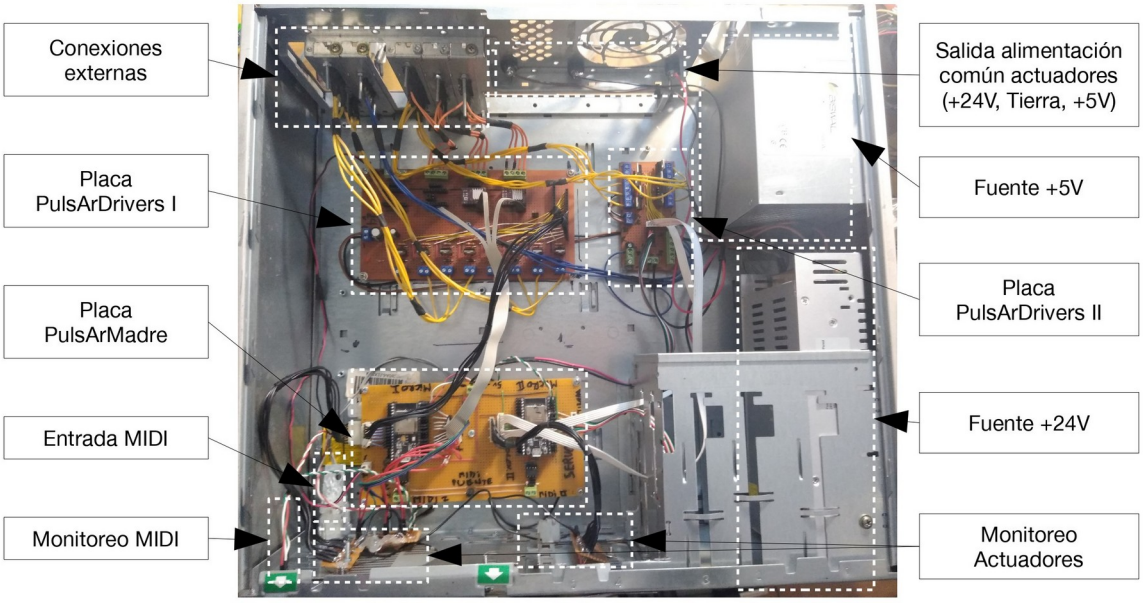

Fig. 44: Detalle Gabinete con componentes internos

Se debió remover una parte del chasis dispuesto para alojamiento de discos rígidos, unidades floppy y ópticas, para ganar espacio para las placas y conexionado. Esto se puede apreciar comparando las últimas 2 figuras.

Para la disposición de la entrada MIDI (DIN de 5 pines a 180º hembra) se debió utilizar una ficha para cable por no conseguir ficha para chasis ni para placa[51]. Esta se montó sobre el chasis del gabinete adhiriéndola a un angulo de aluminio con silicona y ajustando este con tornillos. A su vez, la sujeción de las placas al gabinete se realizó con separadores de placa plásticos roscados y tornillos.

Para disponer las salidas hacia los actuadores se utilizaron borneras con tornillo robustas, colocadas en la zona de conexiones a placas del gabinete. Para cerrar el panel frontal se utilizó chapa recuperada de la fuente ATX original del gabinete, la cual se encontraba inservible.

Para el conexionado de datos entre placas se utilizaron viejos cables planos recuperados de múltiples dispositivos conectados a las terminales de pines machos construidas; a su vez, para las conexiones hacia las terminales exteriores se utilizó cable de mayor grosor, recuperado de múltiples fuentes ATX desarmadas, aprovechando también su variedad de colores para establecer un código de color interno, dentro del gabinete:

- Negro: Tierra
- Rojo: +5V
- Marrón: +24V
- Amarillo: Solenoides

51 La mencionada ficha DIN de 5 pines a 180º, tradicional terminal para los puertos MIDI esta completamente deprecada en otras aplicaciones por lo que prácticamente no se logra adquirir en proveedores de componentes, lo cual plantea un problema para la fabricación de dispositivos DIY con implementación de MIDI.

- Naranja: PAPs
- Azul: Servos.

Finalmente, se utilizó un sistema de bornes de alimentación común, de +5V (para servos) y +24V para solenoides, sobre el panel exterior mediante tornillos, los cuales a futuro deberán ser reemplazados por una opción similar pero más resguardada, sin sobresalir de la estructura del gabinete.

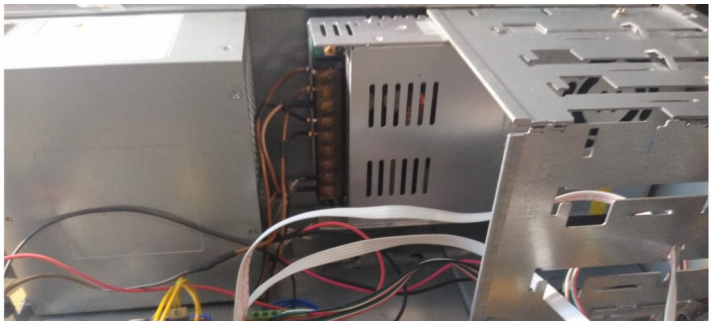

Fig. 45: Detalle montaje de las fuentes de alimentación en el gabinete

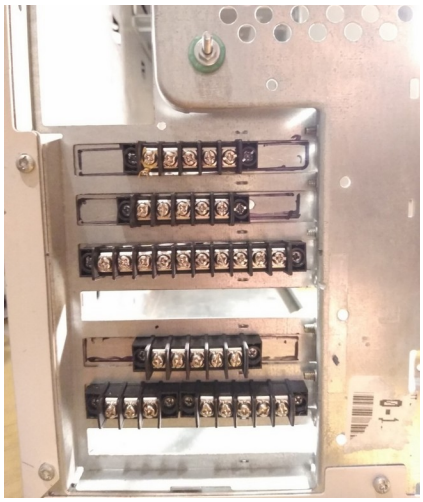

Fig. 46: Detalle Salidas a actuadores

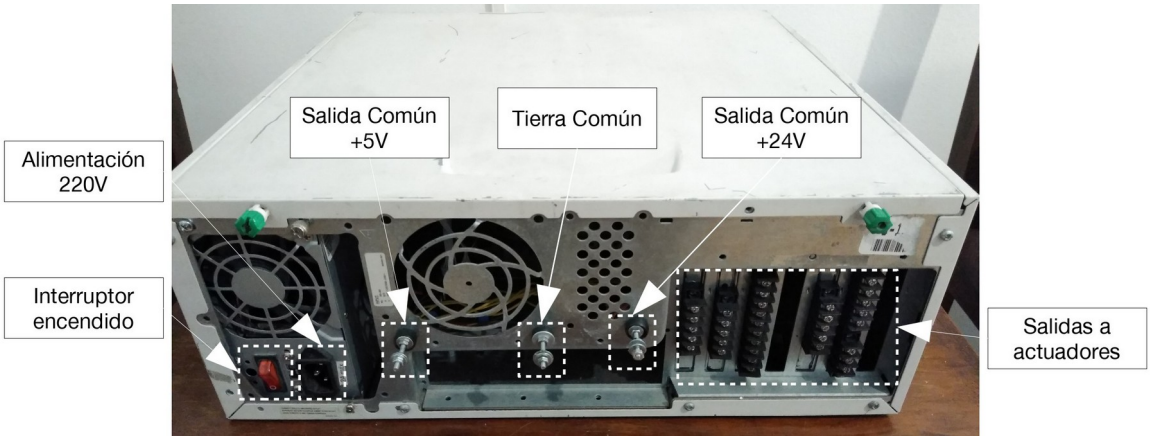

Fig. 47: Detalle vista trasera del gabinete PulsAr-CC

3.2.8 Conexionado gabinete-motores

Por último, en lo que respecta al cableado desde el gabinete hacia los respectivos motores, se utilizó un cable de cobre bipolar de 0,5mm con aislación de PVC transparente. A las puntas de las líneas a conectar a las borneras se les adhirió terminales en U de cobre, con una terminación con termocontraible, y puntas estañadas a las líneas destinadas a los servos.

3.2.9 Resultados obtenidos en fase definitiva

- Diseño de placas que cumplió con las expectativas planteadas, con el plus de contar con la arquitectura hackeable enunciada antes en la placa madre.
- Solución robusta y con posibilidad de transporte para concentrar la electrónica, con un sistema de conexión a motores que combina un equilibrio adecuado entre facilidad y rapidez de conexión y costo no tan elevado.
- Sistema de monitoreo de LED eficiente, tanto en lo que respecta a la recepción de señal MIDI como a la actividad de los MC de cara a los drivers de motor.
- Estructura interna de interconexionado relativamente prolija gracias al uso de cables planos y mangueras recuperados, código de color, etc.

4. Dispositivos electromecánicos

Se contemplan aquí los instrumentos creados: percutores / excitadores y modificadores de cuerpos sonoros. Al igual que con los demás aspectos del proyecto la construcción se observa aquí fases de prototipado y definitiva.

4.1. Construcción Fase prototipo

Fig. 48: Percutores PulsAr en sus versiones prototipo

El objetivo de los prototipos construidos fue establecer las características de funcionamiento, problemas y desafíos en la construcción, pros y contras de cada motor y estrategia de construcción utilizada. Se realizaron:

- Cuatro percutores:
  - Dos de ellos, basados en motores solenoides lineales
  - Uno, basado en motor rotativo de corriente continua combinado con un micro-servo
  - Y uno, basado en un motor paso a paso.
- Un agitador (shaker) de granos basado en un motor paso a paso

4.1.1 PulsAr-PS: Percutores basados en solenoides lineales

Los percutores basados en motores solenoides lineales construidos tienen el nombre de PulsAr-PS, junto con un número de serie único. fueron inspirados en el percutor *KalTron*[52] (diseñado por Michael Darling) con motor solenoide lineal con pivote trasero.

52 Utilizado en el instrumento Karmetik Notomoton de Ajay Kapur, Michael Darling, Jim Murphy, Dimitri Diakopoulos, Jordan Hochenbaum. 2010. https://www.karmetik.com/karmetik-robotics/notomoton

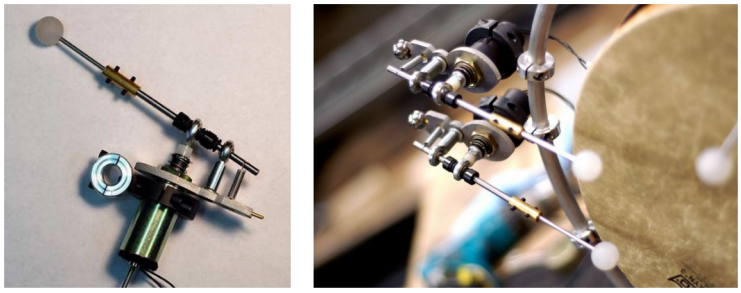

Fig. 49: Percutor KalTron de M. Darling.

Los mismos consisten en una varilla percutora móvil sostenida por un pivote trasero que funciona como vértice fijo, y accionada en un movimiento angular por un solenoide de tirado (*pull*) que, al accionarse, mueve la varilla hacia abajo, donde se encuentra el cuerpo a percutir.

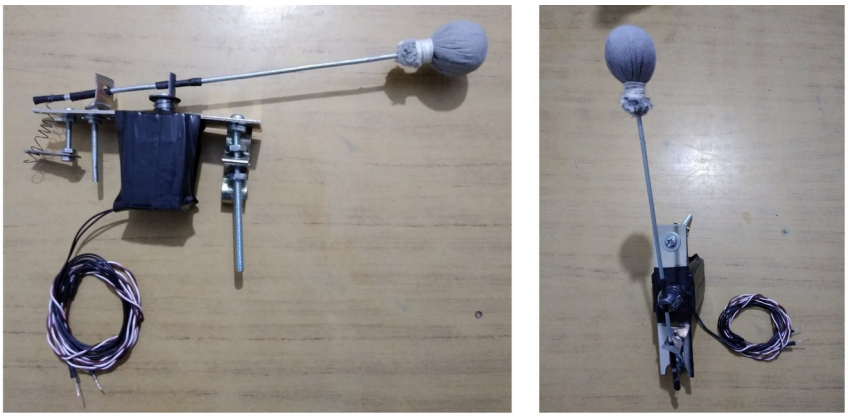

Fig. 50: El percutor prototipo PulsAr-PS I

PulsAr-PS I y PulsAr-PS II fueron versiones prototipo de este diseño de percutor. PS I fue construido utilizando un perfil metálico recuperado de una cerradura, junto con metales varios obtenidos del material recuperado de los artefactos desarmados. El motor fue adherido a la placa estructural mediante cinta aisladora. Se usó alambre galvanizado para la varilla percutora, tubo termocontraible como amortiguador de rose de las partes móviles. Y en el extremo de la varilla que debe entrar en contacto percutir se utilizó una esfera de madera recubierta con una capa fina de algodón, forrada con tela fina, también de algodón.

Se utilizó un resorte trasero para favorecer el retorno de la varilla percutora a su posición de en descanso ajustado a una de altura variable para calibrar la tensión del mismo. Finalmente el mecanismo de sujeción a un tambor (clamp) se construyó usando piezas metálicas recuperadas perforadas y dobladas.

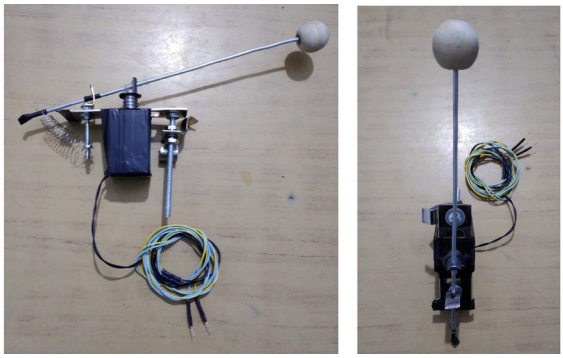

Fig. 51: PulsAr-PS II

EL PulsAr-PS II tiene una estructura prácticamente gemela respecto de su antecesor. Se utilizó una distancia más corta entre el solenoide y el pivote trasero y se dejó la esfera de madera sin recubrimiento externo.

4.1.2 PulsAr-PRS: Percutores basados en combinación de motores

El PulsAr-PRS es un percutor basado en motor rotativo de corriente continua combinado con un micro-servo, inspirado en el percutor Nudge de Murphy, Carnegie y Kapur basado en 3 motores, del cual se tomó la idea de incluir un mecanismo para el control de la posición de "en descanso" (on-rest) del percutor en función del control preciso de la relación intensidad / velocidad de repetición, basado en un motor servo.

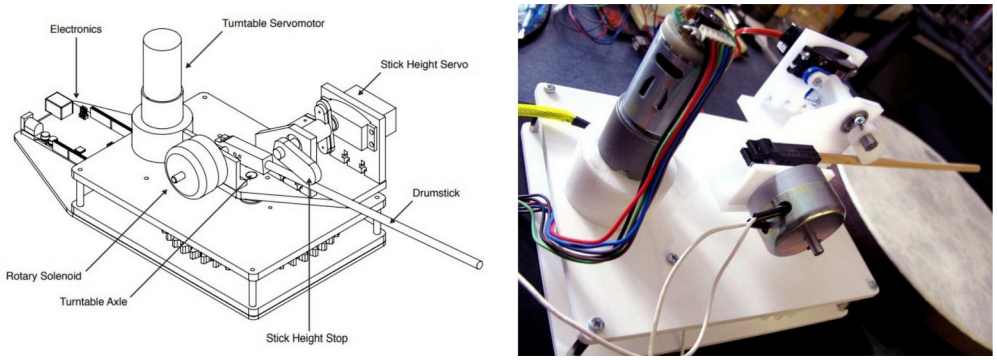

Fig. 52: Percutor Nudge, construido con múltiples actuadores

El percutor construido posee varios puntos de similitud con los PulsAr-PS en su base constructiva. El núcleo de acción se lo da un motor de CC con su eje dispuesto de manera perpendicular al percutor, al cual está fijada la varilla / brazo del dispositivo. En el lado opuesto, posee un micro-servo el cual permite el control de la posición de en descanso de la varilla percutora. Como mecanismo de retracción en este modelo se utilizó una banda de goma, la cual sería reemplazada en la versión definitiva dada su mala performance.

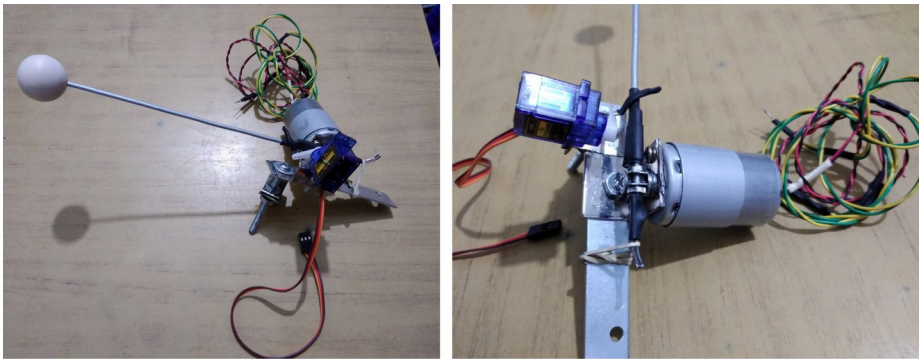

Fig. 53: PulsAr-PRS I

4.1.3 PulsAr-PR

Variante simplificada del dispositivo anterior es PulsAr-PR, percutor construido utilizando solamente un motor rotativo de CC, incorporando un mecanismo de retraimiento, alcanzando un funcionamiento prácticamente idéntico a los motores solenoide rotativos.

Aun con disposiciones y tamaños diversos, este modelo de percutor resulta muy versátil por relativamente sencilla construcción

4.1.4 PulsAr-PPAP: Percutor basado en motor PAP

Finalmente, el último de los percutores prototipados fue el PulsAr-PPAP, con una lógica de acción similar al percutor basado en motor rotativo de CC, solo que en este caso se utilizó un motor PAP bipolar. Dadas las capacidades de control relativamente preciso de la posición y de retención, no se utilizó mecanismo de retracción como resorte o similar. Sin embargo, si debió usarse un mecanismo de calibración de posición basado en final de carrera, descrito anteriormente en el apartado de programación.

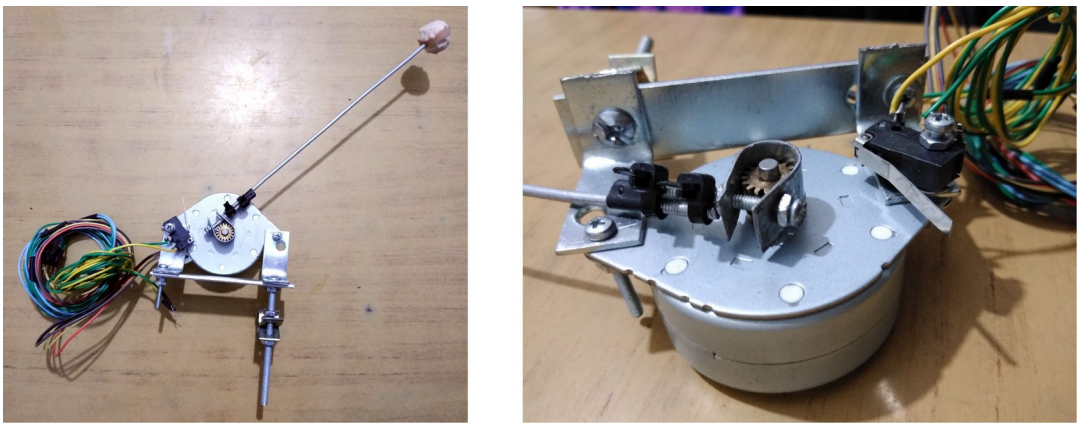

Fig. 54: PulsAr-PPAP

Se esperaba con este actuador lograr no solo funciones de percutor sino también de atenuador (*damper*) pudiendo ejercer una presión controlada en la superficie a excitar (por ejemplo, el parche de un tambor).

Los resultados obtenidos en las pruebas realizadas con este dispositivo evidenciaron su capacidad de ejercer una velocidad y presión de excitación sobre el cuerpo a percutir, consiguiendo mayor intensidad sonora. En el mismo sentido se consiguió el efecto deseado de posibilidad atenuación de parche pudiendo realizar algunos matices en la ejecución con el resto de los percutores.

Sin embargo, se evidenció un problema crítico frente a la pérdida de pasos del motor PAP y por consiguiente la necesidad permanente de re-calibración. Dicho problema intentó resolverse de múltiples formas: modificando la configuración de micro-paso del driver, modificando la programación, ajustando el ángulo de percusión, etc. Sin embargo el problema se mantuvo, lo cual llevó a abandonar este modelo de percutor, proyectando incluir en los desarrollos definitivos un servomotor que pudiera accionar sobre los parches de tambor para lograr el efecto de atenuación y control de tensión deseado.

4.1.5 PulsAr-Ag: Agitador basado en motor PAP

El agitador PulsAr-Ag es un dispositivo construido utilizando un motor PAP bipolar a cuyo eje rotativo se le fijó una varilla roscada la cual incorpora en la punta un pequeño recipiente plástico destinado a contener granos. Mediante la acción controlada de giro del motor se busca agitar las partículas contenidas en el recipiente mencionado obteniendo un efecto similar a una maraca o *shaker*.

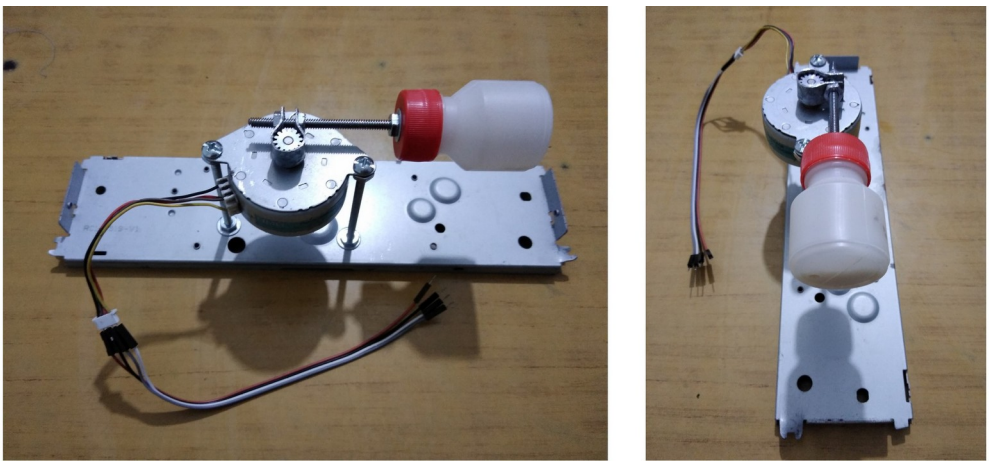

Fig. 55: PulsAr-Ag I

Las pruebas realizadas con este dispositivo fueron muy satisfactorias logrando una gran capacidad de control parametrizable del cuerpo sonoro y por tanto proyectando una buena capacidad expresiva.

4.1.6 Otras estrategias descartadas

Cabe mencionar, para finalizar la descripción de la fase prototípica, que fueron testeados otros mecanismos de generación de sonido, los cuales no llegaron a la fase de terminación de prototipo funcional. Estos son:

- Sistema de carro móvil sobre riel, basado en el sistema de movimiento de cabezal extraído de una impresora chorro a tinta.
- Excitación de placas y cuerdas mediante bobinas construidas con buzzers recuperados de artefactos electrónicos como teléfonos, gabinetes de computadora etc.

En cuanto al primero, las limitaciones en cuanto a posibilidad de calibración, mayor dificultad de montaje y control y una relación de costo/beneficio deficitaria respecto de los resultados sonoros alcanzables llevó a abandonar esta línea de trabajo.

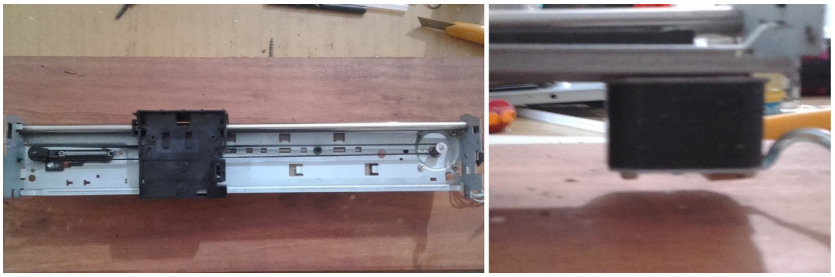

Fig. 56: Pruebas con carro sobre riel

Respecto a la estrategia de acción mediante bobinas con núcleo ferroso (en una palabra, electroimanes), el trabajo con buzzers de distinto tipo redundó en material sonoro de enorme interés tímbrico pudiendo excitar los diferente modos / parciales armónicos de una cuerda o incluso placa metálica mediante la generación de tonos de diferente frecuencia. Como contrapartida la intensidad sonora conseguida fue muy baja. Se intentó trabajar con electroimanes de mayor potencia, pero la dificultad para conseguirlos derivó en un abandono de esta técnica.

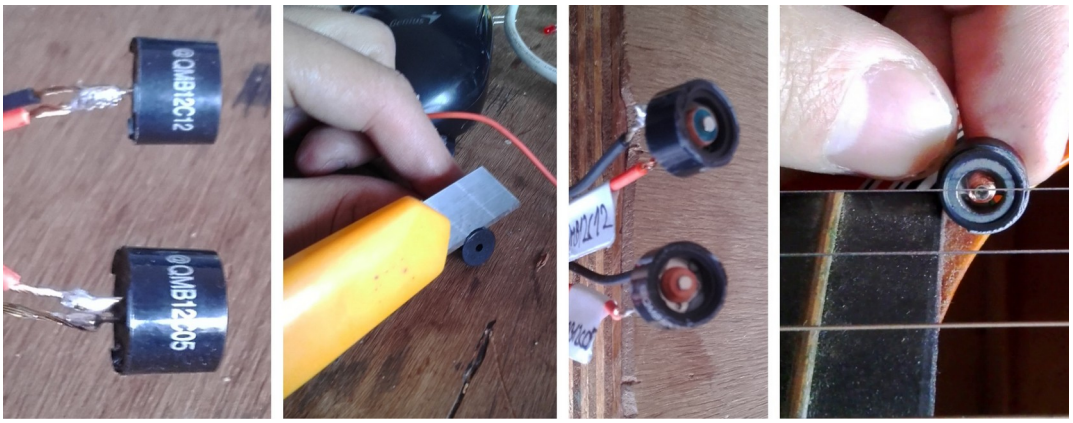

Fig. 57: Pruebas con buzzers

4.1.7 Registro audiovisual de fase prototipo

- Primer prototipo percutor: https://www.youtube.com/watch?v=fQBNbg8SvSA
- Primer prototipo percutor mejorado, usando MIDI en tiempo real: https://www.youtube.com/watch?v=_tcPtoNOlZI
- Prototipos percutores 1 a 4 ejecutando secuencia MIDI compleja: https://www.youtube.com/watch?v=UGkCh2SyUDw
- Prototipos percutores 1 a 3 y prototipo shaker ejecutando secuencia MIDI compleja: https://www.youtube.com/watch?v=yRjPXieuSGY

4.1.8 Resultados obtenidos de fase prototipo

- Los percutores basados en solenoide lineal obtuvieron el mejor equilibrio entre velocidad de repetición de golpes, intensidad sonora, consistencia en cuanto a calibración, etc., en esta fase de desarrollo. Se evidenciaron respuestas más consistentes manteniendo distancias más cortas entre el motor y la punta del percutor.
- Necesidad de abandonar cualquier material rudimentario o poco robusto como cinta adhesiva, pegamento, etc., exceptuando el termocontraible que mostró ser un recurso provechoso para detalles de terminación, atenuador para rozamientos, etc.
- Si bien es utilizada en artefactos comerciales, se evidencia problemático el uso de goma como mecanismo de retracción, por su sensibilidad a la humedad, temperatura, radiación solar. En su lugar, se deben usar resortes metálicos.
- El percutor PulsAr-PRS mostró una respuesta aceptable en las pruebas prototipo, aunque inferior a su hermano PS. Aun así se mantiene el modelo para la versión definitiva[53]. Resulta particularmente interesante una respuesta tipo rebote del percutor frente al impacto contra el parche, producida tanto por la forma particular de acción del motor rotativo de CC como por la elasticidad ofrecida por el brazo metálico construido con alambre galvanizado como. Esta característica se vería atenuada en buena medida en las versiones definitivas en las que se optó por utilizar varillas cilíndricas de madera para el brazo.
- La utilización de Servos Sg90 mostró resultados poco satisfactorios. Por su frágil estructura basada en engranajes plásticos, una exigencia mayor sobre el motor provocó la rotura de varias unidades. A fin de prevenir esto se optaría, como se mencionó antes, por utilizar brazos de madera en el percutor, a fin de aminorar el efecto rebote y reducir la fuerza del impacto sobre el eje del Servo. Así también se evidenció la necesidad de incluir un mecanismo de calibración de posición mínima y máxima de la acción del motor (tal como se describiera en la sección de control lógico), junto con un sistema de posicionamiento inicial a 90º (mitad del recorrido), para evitar forzar el motor previo al momento de calibración.

53 Sin embargo, los modelos definitivos creados, utilizando madera en vez de alambre para el brazo del percutor, mostraron una respuesta altamente superior a las versiones prototipo, teniendo uno de los modelos incluso una performance superior a las versiones finales basadas en solenoides lineales.

- El percutor PulsAr-PPAP obtuvo las más altas intensidades sonoras en el golpe. Lamentablemente, por su permanente descalibración, complejidad de uso, etc., se optó por descartarlo.
- Un interesante efecto de interacción humano-máquina durante la ejecución fue tensar el parche presionando con el dedo en el centro del mismo, logrando una elevación de su afinación. En función de esto, se buscaría utilizar, en las versiones definitivas, un servomotor de mayor torque y con engranajes metálicos (Mg996) para lograr vencer la resistencia del parche sin dañar el motor.
- Las pruebas con PulsAr-Agit fueron, como se dijera, muy satisfactorias. Se evidenció sí la necesidad de buscar un mecanismo de sujeción alternativo a las placas para sobre mesa, ya que estas redundaban en gran cantidad de ruido adicional indeseable frente a la vibración del actuador.
- Por último, sobre la paleta sonora conseguida, se evidenció la necesidad de incorporar sonidos de tipo metálico (tipo cencerro), sonidos con sostenimiento (*sustain*, tipo platillo o placas), sonidos de percusión en madera y sonidos de más baja frecuencia, ya sea de parche o de placa metálica. En función de esto se proyectarían los desarrollos para la versión definitiva.

4.2. Construcción Fase definitiva

Para la fase definitiva, se avanzó en la idea de construcción de instrumentos completos, más allá de percutores individuales o mecanismos aislados. Es en este punto donde el conjunto del sistema muestra unidad de funcionamiento. Sobre la base de los conceptos de robustez y durabilidad tematizados por Raes, se utilizó mayormente metal, por su durabilidad y solidez estructural. Así mismo, se buscó construir estructuras para el soporte de percutores y demás partes de los instrumentos.

4.2.1 PulsAr TaboT I y II

Se trata de dos tom-toms recuperados, los cuales fueron restaurados y sobre los que se construyó un dispositivo externo metálico de caño estructural de 20mm de grosor y 2mm de espesor del metal para la colocación calibrada de percutores. A su vez, se incluyó un soporte interno para la fijación del sistema de atenuación y control de tensión del parche.

Fig. 58: PulsAr-TaboT I y II

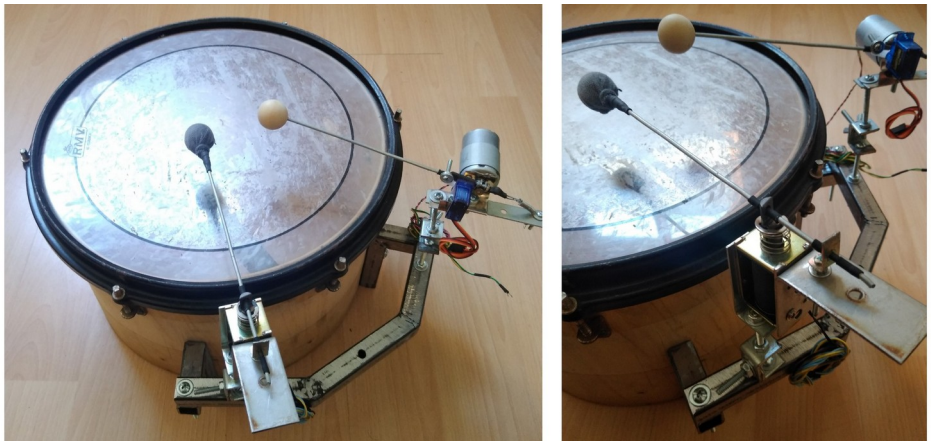

Fig. 59: PulsAr-TaboT I

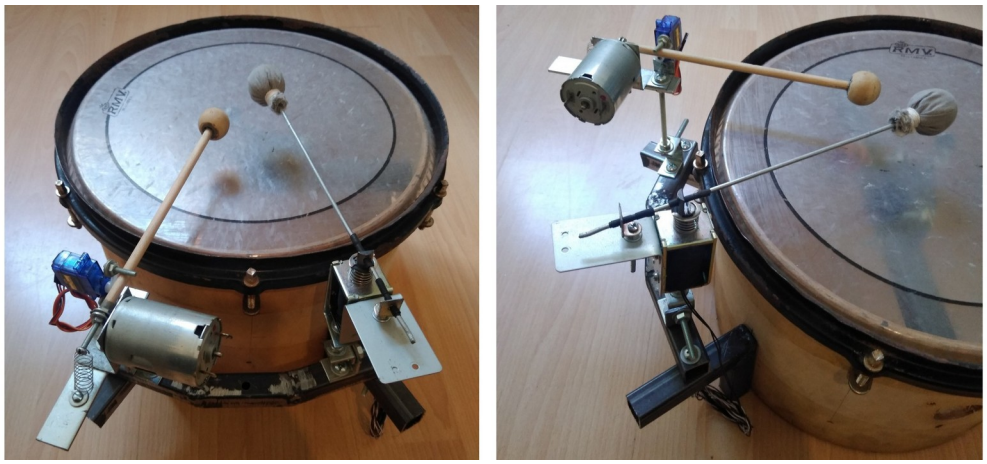

Fig. 60: PulsAr-TaboT II

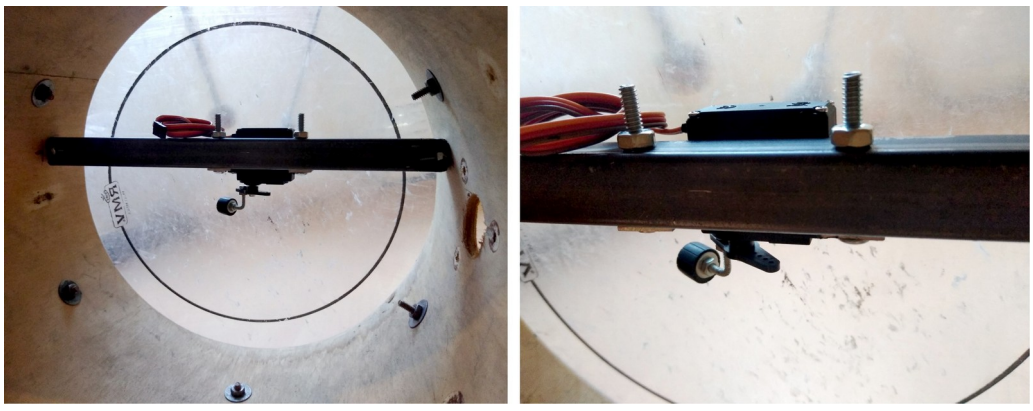

Fig. 61: PulsAr-TaboT. Detalle mecanismo interno de atenuación y control de tensión de parche.

Características

- **Cuerpo sonoro:** Se utilizaron 2 tom-toms de batería de 12 y 13 pulgadas respectivamente, construidos con madera terciada, aros metálicos, 6 torres, y parches hidráulicos solo superiores.
- **Actuadores:** Se utilizaron 2 percutores en cada tambor, un PulsAr-PS (modelos III y IV respectivamente) y PulsAr-PRS (también modelos III y IV). A su vez se incluyó un atenuador/controlador de tensión en la parte interna del instrumento basado en servomotor Mg946r.
- **Marco metálico de montaje de motores y soporte:** Por otro lado, se construyó un marco metálico para montaje de motores y como soporte del instrumento con caño estructural cuadrado de 20mm.
- **Conexionado:** El conexionado se realizó usando cable de 0.5 mm.

4.2.2 PulsAr-AgitR

Se trata de 2 un dúo de PulsAr-Ags, agitadores tipo shakers basados en motor PAP, construidos en la fase de prototipo.

Fig. 62: PulsAr-TaboT. Detalle mecanismo interno de atenuación y control de tensión de parche.

4.2.2.1 Características

- **Cuerpo sonoro:** Se utilizaron frascos plásticos rellenos de granos de arroz fino en uno y de lentejas en otro, a fin de conseguir diferentes timbres al momento de la agitación.
- **Actuadores:** Se utilizaron motores PAP bipolares PM55L extraídos de máquinas fotocopiadoras.

- **Marco metálico de montaje de motores y soporte:** El soporte se construyó utilizando tornillos con tuerca de 4mm para su fijación al soporte de caño estructural de 20mm, junto con un tornillo y tuerca de 6mm para fijación a trípode de fotografía.
- **Conexionado:** El conexionado se realizó usando cable bipolar de 0.5 mm. utilizando 2 pares de hilos por cada motor (correspondientes a sendas 2 boninas), para un total de 8 líneas.

4.2.3 PulsAr-LatX

Consiste en 3 latas de conserva de diverso tamaño, percutidas por mini-solenoides con acción angular a los que se les ha agregado un tornillo como elemento percutor.

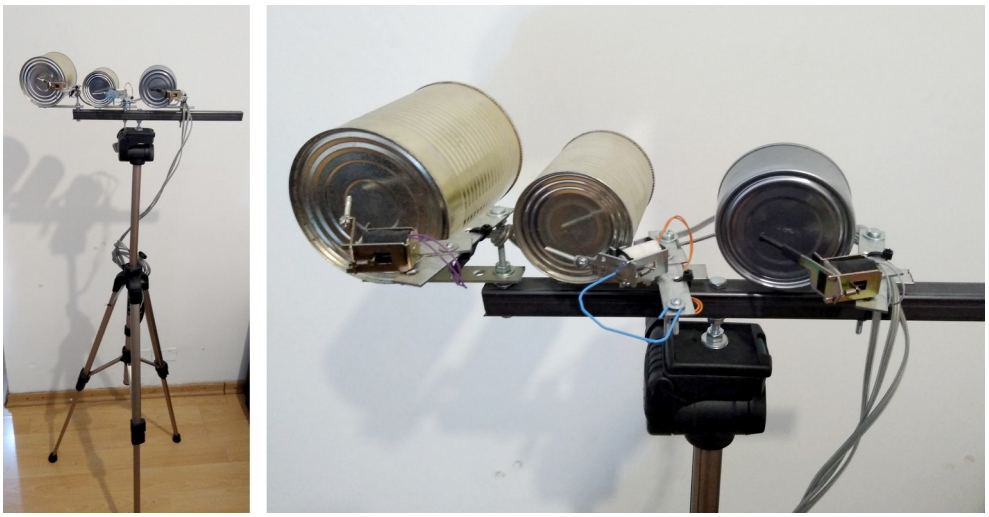

Fig. 63: PulsAr-LatX

4.2.3.1 Características

- **Cuerpo sonoro:** 3 latas de 10x12cm, 7,5x9,5cm y 8x4cm respectivamente.
- **Actuadores:** Mini-solenoides de acción angular.
- **Marco metálico de montaje de motores y soporte:** Se utilizaron placas metálicas de 1,5mm de espesor para la estructura de montaje y calibrado del motor y percutor y caño estructural de 20mm para el armado del soporte global
- **Conexionado:** se realizó usando cable de 0.5 mm.

4.2.4 PulsAr-SinA

Fig. 64: PulsAr-SinA

Se trata de un platillo recuperado tipo China, quebrado en varias partes, en muy mal estado, sobre el cual inciden 2 actuadores: un PulsAr-PR y un mini-solenoides de acción angular. Cabe mencionar que la disposición del platillo es invertida a la comúnmente utilizada, ya que en las pruebas de funcionamiento se evidenció así un sonido más interesante.

- **Cuerpo sonoro:** Platillo tipo China Zildjian Abedis de 18" dispuesto en forma invertida
- **Actuadores:** PulsAr-PR con calibrador estático de posición de *en-descanso*; mini-solenoide de acción angular con terminación en termocontraible a modo de amortiguador de impacto.
- **Marco metálico de montaje de motores y soporte:** Se utilizó la base de un atril de aluminio recuperado, tornillos de 4 y 6mm para las diversas sujeciones y diversas placas metálicas de 2mm de espesor para el montaje de actuadores.
- **Conexionado:** se realizó usando cable de 0.5 mm.

4.2.5 PulsAr-PuQ

Fig. 65: PulsAr-PuQ

Consiste en unos bongos, partidos en su base, sobre los que se montó un marco metálico para el montaje de actuadores. La base del instrumento se encuentra disponible y se proyecta restaurar a futuro.

4.2.5.1 Características

- **Cuerpo sonoro:** Bongos de madera marca Meinl,
- **Actuadores:** 2 mini-solenoides de acción angular, cada uno con un tornillo extensor de su parte móvil y un pequeño aplique de madera en la punta del mismo para amortiguar el impacto.
- **Marco metálico de montaje de motores y soporte:** Tornillos de 4 y 6mm para las diversas sujeciones y una placa de acero inoxidable de 1.5mm de espesor junto con otra de aluminio transversal (recuperada del material estructural de impresoras) para el montaje de motores.
- **Conexionado:** se realizó usando cable de 0.5 mm.

4.2.6 PulsAr-BloM

Consiste en un bombo de batería recuperado, al que se le incorporó un percusor tipo PulsAr-PR, basado en un motor NA4565D de la empresa Nisca, usado originalmente como actuador en un elevador de fotocopiadora Canon. Las mayores dimensiones del instrumento en comparación al resto de los dispositivos creados, plantearon un aumento de escala importante en todos sus aspectos constructivos.

El motor posee una potencia más elevada que el resto de los motores, lo cual plantea diversas problemáticas específicas para su operación. En primer lugar, debió utilizarse varilla roscada de 3mm junto con un segmento de baqueta de batería para la construcción del brazo percutor, ensayos con otros materiales resultaban en inmediatas roturas.

Por otro lado, y como aspecto muy importante, los drivers basados en TIP122 no son aptos para manejar los más de 5A que puede consumir el actuador. Pruebas iniciales quemaron uno de los canales de driver de solenoide de la placa PulsAr-Drivers1.

Ensayos con múltiples configuraciones arrojaron que la conexión a la misma fuente de 24V usada en el gabinete redunda en una potencial sobrecarga del sistema, por lo que se optó por utilizar una fuente ATX específica (la empleada en la fase prototipo) con su salida de 12V (la cual mostró intensidades de operación igualmente satisfactorias a pesar de estar al 50% de la tensión de trabajo máxima admitida por el motor) y un Relé 2PH63091A con el que se contaba previamente, como driver. A este respecto se debió utilizar un convertidor de niveles lógicos de 3.3V a 5V tipo DEV_0347, para poder operar el relé con el ESP32. La arquitectura hackeable de la PulsAr-Madre jugó un papel en poder redireccionar la misma señal generada para el módulo de driver solenoide averiado de la PulsAr-Drivers1 hacia el convertidor de niveles y de allí al relé.

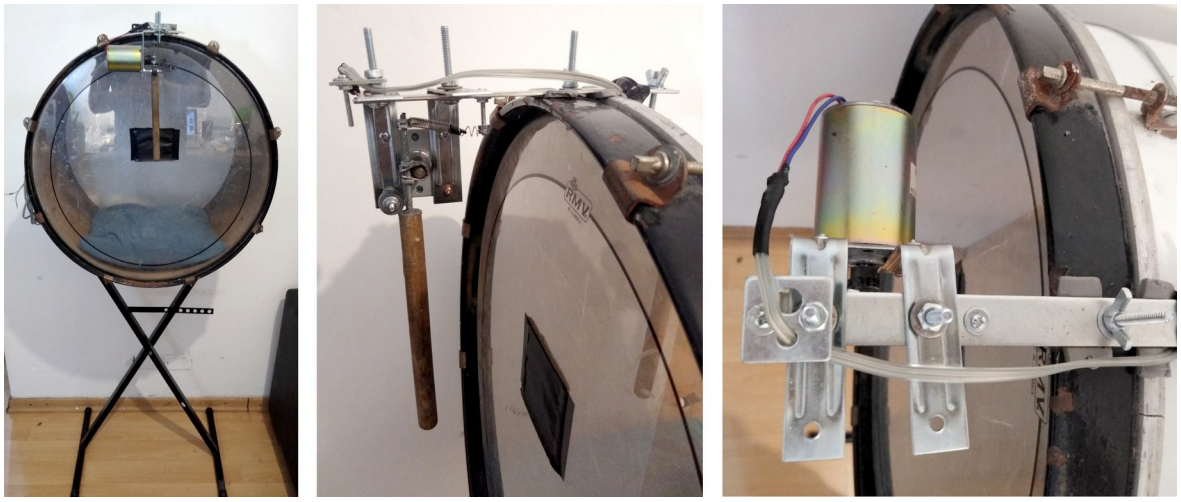

Fig. 66: PulsAr-BloM

4.2.6.1 Características

- **Cuerpo sonoro:** Bombo de batería de 22" recuperado, con parche frontal únicamente marca RMV.
- **Actuadores:** Percutor tipo PulsAr-PR basado en motor Nisca NA4565D de 24V.
- **Marco metálico de montaje de motores y soporte:** Tornillos de 2, 4 y 6mm para las diversas sujeciones y una placa de acero inoxidable de 2mm de espesor junto con ángulos de 1.5mm, para el montaje de motores.
- **Conexionado:** se realizó usando cable de 0.5 mm.

4.2.7 Resultados obtenidos en fase definitiva

- Se obtuvieron resultados mejores en el trabajo con motores rotativos de CC, utilizándolos prácticamente como solenoides, que los conseguidos durante la fase prototipo.

- En su implementación en percutores, el punto de mayor complejidad radica en la fijación de la varilla percutora al rotor. La técnica puesta en práctica resuelve relativamente el problema, pero al tratarse de un mecanismo de sujeción por compresión es más plausible de ser desajustado en el uso que mecanismos con trabas o pasantes.
- Es fundamental, la realización de pruebas sistemáticas de funcionamiento, respecto de la posición de los percutores, tensión de resortes, etc. para dar cuenta de las mejores performances.
- El aplique utilizado en el servo de ajuste de tensión de los parches en los PulsAr-TaboT es un punto débil en la construcción. Si bien su efecto sonoro es satisfactorio, resulta poco robusto en tanto que se desajusta recurrentemente. Se evalúa su posible eliminación a futuro, haciendo que el servo accione de manera directa sobre el parche, debiendo acercar el motor a este.
- Resulto un acierto la solución estandar de utilizar trípodes de fotografía como soportes de los dispositivos más pequeños y livianos, inspirada en el trabajo de Moritz Simon Geist con sus Sonic Robots.
- Se evidencia la necesidad de perfeccionar el mecanismo de cableado de los instrumentos hacia el gabinete. El cable utilizado es efectivamente muy robusto, y adecuado para conexiones bipolares simples. Pero se vuelve engorroso al incorporar múltiples líneas hacia un mismo instrumento. En este punto sería ideal contar con un solo cable multi-filamento resistente. A su vez, a futuro será importante mejorar el mecanismo de sujeción de cables a los instrumentos, para evitar desconexiones o "tironeos" que puedan cortar el circuito.

Conclusiones y perspectivas

Se enumeran a continuación las conclusiones alcanzadas en el transcurso del proyecto.

En primer lugar, respecto a la planificación, comparando plan original y proceso real de trabajo, se advierte que este tipo de proyectos de construcción de dispositivos necesariamente debe incorporar como una fase fundamental y de largo aliento el momento de prototipado. Es en dicho momento donde se contrastan las hipótesis inicialmente planteadas con la práctica y donde se puede proyectar las soluciones definitivas que se podrán comprobar finalmente en el proceso de construcción definitiva.

En segundo lugar, sobre el uso de materiales recuperados, sin dudas es una estrategia de abastecimiento de materiales que repercute en un abaratamiento de costos. Sin embargo, se debe atender al tiempo de desarme, extracción y recuperación de material útil el cual no es en absoluto despreciable y debe ser contemplado como un costo a la hora de evaluar los presupuestos.

La dificultad para encontrar especificaciones técnicas de los motores, el funcionamiento no siempre consistente dado el diferente desgaste a cuesta que tenga cada aparato, etc. repercuten de manera notable en una gran dificultad para lograr precisión y consistencia en la acción de los dispositivos. Entonces, no es mediante desarrollos con este tipo de material con los que se va a lograr una respuesta cinética con agudo control parametrizable: *el uso de materiales recuperados reduce la precisión*, hablando en términos generales.

Sin embargo, esto para nada significa que con dichos materiales no se pueda conseguir resultados sonoros interesantes. Simplemente se debe ajustar la estrategia de abordaje en la construcción de objetos sonoros: *se debe apuntar más a un abordaje extensivo*. Es decir, por ejemplo, que para lograr mayor capacidad expresiva, riqueza tímbrica, etc. se deben incluir más objetos sonoros, en donde la intervención del actuador sea más acotada y simple, en vez de un abordaje intensivo, el cual implique una respuesta lineal controlada en todo el espectro parametrizable de los actuadores. Esto también como noción o criterio general, sujeto por supuesto a particularidades, excepciones, etc.

A su vez, es interesante incorporar aquí el concepto de "*antiluthería*" de Laxague[54]. Antes que tratar de amoldar de manera rígida los materiales recuperados a diseños mecatrónicos predefinidos, suele ser más conveniente partir de los materiales obtenidos y tomar su forma, condiciones, capacidades, cantidad y variedad, y diseñar con ellos instrumentos concretos particulares, los cuales se adapten a dichas características.

Finalmente, cabe decir que el trabajo con materiales recuperados redunda en enorme contenido y experiencia de aprendizaje, ejercitando el ingenio[55]. Es una experiencia con fuerte

54 Este propone, para la creación de instrumentos acústicos de madera, invertir la ecuación tradicional, en la que se busca el material en función de un diseño definido a priori. En este punto, Laxague busca construir instrumentos con materiales recuperados, en los cuales el diseño definitivo de los mismos esté dado principalmente por la forma y tipo de material, a los cuales los diseños tradicionales deben amoldarse. Ver "*Tata Laxague: Antilutheria electroacústica, «entre la magia y la ciencia»*" (https://www.republica.com.uy/tata-laxague-antilutheria-electroacustica-entre-la-magia-y-la-ciencia-id703948/)

55 Y el "retro-ingenio", a partir de la comprensión del funcionamiento molecular de los dispositivos complejos desde el todo a la parte, mediante su desarme, su "hackeo".

contenido lúdico, una suerte de gran rompecabezas (sin dudas con reminiscencias al antiguo mecano), en el que tornillos, tuercas y chapas se combinan para la construcción de objetos artísticos. Combinado con una adecuada incorporación de teoría general resulta en una muy rica estrategia de aprendizaje en el campo de la mecatrónica.

En tercer lugar, en lo que respecta al desarrollo del sistema de control lógico, fue un acierto el pasaje a un enfoque tipo POO. Esto favoreció la modularidad y el bajo acoplamiento de los bloques del programa, aspectos importantes para su mantenimiento y reutilización.

A su vez, resultó acertado la implementación temprana del sistema MIDI como mecanismo de control, incluso desde los inicios del prototipado. Esto proveyó una solución definitiva para el manejo de la actuación de los dispositivos *desde un punto de vista musical*, es decir, integrada a controladores MIDI, secuenciadores, etc.

Así mismo, redundó en un flujo de trabajo mucho más productivo la utilización de un sistema de control de versiones profesional (Git), junto con una IDE más robusta en comparación a la provista por Arduino.

En cuarto lugar, en cuanto al desarrollo electrónico, resultó de gran ayuda la fabricación de placas de desarrollo en la etapa de prototipado a fin de despejar parte del trabajo de corrección de errores, los cuales se vuelven recurrentes a la hora de trabajo con protoboard.

Fue sin dudas un acierto la utilización de drivers del proyecto RepRap para los PAP en tanto que ofrecen una solución completa, "todo en uno", poseen bajo costo, garantizan una excelente performance gracias a la implementación de la técnica micro-paso, y son más fácilmente controlables desde el MC.

En el mismo sentido, el resolver buena parte de la alimentación eléctrica mediante una fuente adquirida específicamente ahorró toda una serie de problemas cruciales de funcionamiento del dispositivo, algunos de los cuales si aparecieron en la fase de prototipado y pruebas preliminares utilizando fuentes recuperadas para la alimentación de 5V, junto con un breve lapso de pruebas usando 12V.

Se estima que varios de los circuitos podrían haberse simplificado. El driver de motores solenoide utiliza borneras bipolares en cada modulo, de las cuales solo se utilizó finalmente el punto dinámico del circuito (el que interrumpe o habilita el flujo a tierra), desaprovechando la posibilidad de implementar alimentación positiva común.

En particular la construcción del gabinete mediante un chasis recuperado demandó una gran cantidad trabajo, en cuanto a adaptación, colocación de la electrónica, interconexionado, etc. Este aspecto, subdimensionado originalmente y al que probablemente haya contribuido la falta de herramientas más específicas, debe ser tenido más en cuenta en futuros desarrollos, evaluando uso de otros materiales o estrategias alternativas, etc.

Finalmente, sin dudas un punto a atender sin dudas es que no se han incorporado mecanismos de protección por hardware específicos como fusibles, más allá de los ya existentes en las fuentes y drivers. Así también se deberá evaluar más exhaustivamente conforme el uso el rendimiento térmico del gabinete y la eventual necesidad de incorporar un nuevo disipador, probablemente en la parte frontal del aparato.

En quinto lugar, sobre la construcción de los dispositivos electro-mecánicos, aun habiendo aumentado sensiblemente los tiempos de desarrollo fue un acierto fabricar todos los dispositivos

definitivos en metal con mayor rigidez lo cual deviene en artefactos más perdurables y con mayor estabilidad al momento de utilizarse. Cabe mencionar igualmente que el aumento en el peso y dimensiones de los artefactos es considerable lo cual repercute negativamente en las posibilidades de transporte. Se deberá evaluar el uso de acero o aluminio en futuras construcciones, lo que igualmente complejiza el momento de soldado.

En particular sobre los mecanismos de retracción, los resortes metálicos fueron los únicos que devolvieron una buena performance y perdurabilidad. Si bien la goma es un mecanismo útil y de más fácil calibración se desgasta rápidamente y no posee la misma resiliencia.

En cuanto al diseño de percutores, se advierte que es conveniente no abusar en demasiadas partes móviles en su diseño ya que, si bien ofrecen mayores posibilidades de configuración, no siempre esto repercute en una mejora drástica del material sonoro obtenido y sí deviene en un aumento sensible en los tiempos necesarios de calibración, eventuales desajustes, etc.

Respecto a la utilización de diferentes motores, se obtuvieron respuestas muy satisfactorias tanto de los solenoides lineales como de los rotativos de CC. Al contar sólo con solenoides de tirado (*pull*) la complejidad requerida para su implementación es relativamente igual a la de los rotativos[56].

Respecto al uso de solenoides lineales, hay que evaluar la construcción propia mediante alambre y núcleo ferroso móvil, para el abaratamiento de costos, dado que son dispositivos relativamente fáciles de construir.

Sin dudas, los motores rotativos de corriente continua son los más sencillos de adquirir. Toda impresora chorro a tinta (artefacto que más fácilmente se encuentra como chatarra tecnológica reutilizable para proyectos de mecatrónica aplicada a música) posee 1 o 2 motores de este tipo. Se evidencia en la gran disponibilidad de estos motores en el lote conseguido. En contrapartida, a fin de abaratar costos, las impresoras modernas poseen actuadores cada vez más débiles, lo cual limita su capacidad de acción en tanto percutores implementándolos como solenoides rotativos. Estos motores resultan sí ideales para usarlos con ejes dislocados, como vibradores, para la generación de sonidos estocásticos, raspadores, etc. A su vez, junto con mecanismos de control de dirección de giro con circuitos tipo Puente H y eventualmente usando los DAC de los ESP32 se podría obtener una respuesta altamente, parametrizable comparable a la de los PAP. Mecanismos no exploradores exhaustivamente en este desarrollo y que serán sin duda objeto de futuras pruebas.

Los motores PAP, caracterizados por la posibilidad de control preciso de la posición angular del eje y de su acción de rotación, presentan dificultades considerables para su manipulación, las cuales pueden reducirse si se trabaja con PAP bipolares (en combinación con drivers tipo RepRap). Han demostrado rendir muy bien en dispositivos en los que basta el control de giro, sin importar su posición en la trayectoria angular, siendo los actuadores que mejor se tradujeron en capacidad expresiva del instrumento construido sobre su base. Cabe mencionar también que estos dispositivos aportan una cuota importante de ruido indeseable y

56 En IMRs de percusión, según la investigación realizada, los mecanismos más simples se basan en solenoides de empuje (tipo push), a los que eventualmente se les incluye un complemento simple para modelar el impacto, o incluso pueden utilizarse de forma directa. Ninguno de los motores recuperados responde en rigor a este tipo. Se intentaron conversiones de los actuadores, agujereando el fondo de los mismos, con resultados insatisfactorios. Lo más parecido fueron los micro-solenoides de acción angular, que igualmente incorporan el mecanismo de pivote en su estructura, la cual debe extenderse mediante un tornillo o similar.

vibraciones en la estructura global, lo cual puede ser resuelto en la perfección del soporte, mediantes aislantes de goma y con la técnica de micro-paso.

Finalmente, los motores tipo servo son fácilmente controlables y corren con la ventaja de que no requieren driver. Sin embargo, tienen el problema de la generación de una gran cantidad de ruido acústico no deseado. Resultaron útiles a la hora de realizar movimientos finos de calibración con retención de posición. Sin embargo, la experiencia de uso en este proyecto no fue satisfactoria, por encontrar problemas en la implementación de su mecanismo de control digital junto con otros dispositivos, por la fragilidad constructiva sobre todo del micro-servo, por requerir mayor trabajo de calibración, etc.

Finalmente, en sexto lugar, sobre la utilización del dispositivo para hacer música, los instrumentos poseen una latencia tolerable, que permite su uso en ejecuciones en (quasi) tiempo real y no solo hacer música secuenciada. La optimización de aspectos de la programación y el uso de un MC más poderoso redundó en un mejoramiento de la performance general del sistema que redujo los tiempos de respuesta. La latencia del instrumento es igualmente, por supuesto, superior a la de un instrumento virtual digital y el retardo entre la acción de un controlador MIDI y el ulterior resultado sonoro implica un toque con anticipación[57].

A fin de ganar en expresividad, es clave, a la hora de la traducción de la composición en interpretación, prestar especial atención a un modelado fino de la secuencia, prestando espacial atención a duraciones de notas, pequeños corrimientos anticipatorios necesarios para compensar retardos por la inercia evidente de un sistema físico. Comprender que cuando un ser humano interpreta un instrumento acústico, el comienzo del movimiento que permite un evento sonoro inicia evidentemente antes de que este deba ocurrir en sincronía orgánica con el tactus de la música que se esté produciendo. Si dicho movimiento antediluviano debe tenerse sin dudas en cuenta para cualquier secuencia MIDI más o menos riguroza y delicada, realizada con el rollo de piano (*piano-roll*) en un DAW, destinada a un sampler o sintetizador digital, es mucho más crucial aún que esto sea atendido en secuencias destinadas a ser interpretadas por robots.

Resumiendo, Luego de casi 2 años de trabajo, se concluye que el sistema construido cumple satisfactoriamente con los objetivos generales inicialmente planteados. Se alcanzó un dispositivo de gran aporte en el proceso de aprendizaje e integración de conocimientos, y fundamentalmente un aparato con el que se puede hacer música.

Trabajos futuros

Entre los aspectos a explorar a futuro se encuentran:

- *Pruebas de laboratorio:* Uno de los déficits del proyecto es sin dudas la falta de pruebas sistemáticas y rigurosas que permitan superar las valoraciones subjetivas y de orden cualitativo sobre la respuesta del sistema construido. Esto pretende ser subsanado en lo sucesivo mediante la realización de pruebas de laboratorio, con el instrumental adecuado.
- *Articulación mecatrónica-humano:* explorar las posibilidades de asistencia humana en la ejecución del instrumento construido, fundamentalmente para cuestiones de mayor

57 Esto es algo tematizado por Pat Metheny a propósito de su proyecto Orchestrion, en el cual plantea haber resuelto el problema incorporando en la técnica de ejecución, una mayor anticipación en el toque del dispositivo de control para lograr un resultado sonoro "a tiempo". Ver la entrevista: https://www.soundonsound.com/people/pat-methenys-orchestrion

complejidad de automatización como cambio de baquetas, agregado de sordina o apliques en los materiales, cambio de afinaciones, etc. Se considera entonces que esta articulación mecatrónica-humano es una forma de mejorar la expresividad de un robot musical en el proceso de creación de sonido (Kemper y Barton, 2018).

- *Refuerzo visual*: Si bien el dispositivo ofrece movimientos interesantes para una experiencia sonoro-visual, muchas veces el movimiento de los actuadores es muy sutil y no logra percibirse a simple vista. En este sentido, se podrá explorar la incorporación de estímulos visuales como juegos de luces (creados con LEDs simples o RGB), las cuales pudieran funcionar redundando de forma directa y simple el evento sonoro, o bien ser un elemento expresivo más, con control independiente, para dotar al sistema de mayor capacidad expresiva.
- *Espacialización:* explorar las posibilidades de distribución del sistema en una sala de audiencia, incluso con posibilidad de movimiento asistido o autónomo en el espacio, para sumar esta dimensión como otro parámetro expresivo.
- *Investigación sobre desarrollos de IMRs en Argentina y América Latina:* Resulta de particular interés indagar en profundidad sobre el campo de trabajo en mecatrónica orientada a música de nuestro país y continente
- *Mecatrónica e instrumentos autóctonos y/o relevantes en nuestra cultura:* En línea con lo anterior, y en vistas a contribuir a una línea de investigación con creciente interés a nivel global, es de interés para futuros proyectos el combinar el desarrollo de IMRs con el bagaje instrumental de nuestra tierra.

Octubre de 2019

Bibliografía

- Benavides F., Otegui X., Aguirre A.,& Andrade F. (2013). Robótica Educativa en Uruguay: de la mano del robot Butiá. XV Congreso Internacional de Informática en la Educación, Montevideo, Uruguay.
- Benjamin, Walter (1936). La obra de arte en la era de su reproductibilidad técnica. México: Ítaca
- Bishop, Robert H. (2002). The mechatronics handbook. Boca Raton, Florida: CRC Press
- Buitrago Y., Prieto J., Peña C. (2012). Robot educativo de bajo costo que interpreta melodías en piano Facultad de Ingeniería Universidad Pedagógica y Tecnológica de Colombia.
- Cela Rosero, Andrés Fernando. Escuela politécnica nacional. Quito, Ecuador. 2007. Diseño y construcción de un sistema robótico para tocar una batería de música
- Collins, Nicolas (2009). Handmade Electronic Music: The Art of Hardware Hacking. Routledge
- Cruz Galapito, Héctor Horacio (2016). Diseño y desarrollo de una interfaz física aplicada a la automatización de un instrumento musical de teclado. UNQ, Buenos Aires, Argentina
- Doyle, Tom (2014). Pat Metheny's Orchestrion. Orchestrion Manoeuvres. Sound on Sound.
- Kapur, Ajay (2005). A history of robotic musical instruments.
- Kapur, Ajay (2007). A comparison of solenoid-based strategies for robotic drumming. NIME'15, May 31-June 3, 2015, Louisiana State Univ., Baton Rouge, LA.
- Kapur, Ajay; Darling, Michael; Murphy, Jim; Hochenbaum, Jordan; Diakopoulos, Dimitri; Trimpin. The KarmetiK NotomotoN: A New Breed of Musical Robot for Teaching and Performance NIME'11, 30 May–1 June 2011, Oslo, Norway.
- Kapur Ajay, Murphy Jim, Darling Michael, Heep Eric, Lott Bruce, Morris Ness (2016). MalletOTon and the Modulets: Modular and Extensible Musical Robots. NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.
- Kemper S. y Barton S. (2018). Mechatronic Expression: Reconsidering Expressivity in Music for Robotic Instruments. NIME'18, June 3-6, 2018, Blacksburg, Virginia.
- Long J., Murphy J.W., Kapur A., Carnegie Dale (2015). A Methodology for Evaluating Robotic Striking Mechanisms for Musical Contexts.
- Long Jason, Kapur Ajay, Carnegie Dale A. (2016). The Closed-Loop Robotic Glockenspiel: Improving Musical Robots Using Embedded Musical Information Retrieval. NIME'16, July 11-15, 2016, Griffith University, Brisbane, Australia.
- Maes Laura, Raes Godfried-Willem, Rogers Troy (2011). The Man and Machine Robot Orchestra at Logos. Computer Music Journal, 35:4, pp. 28–48, Winter 2011. Massachusetts Institute of Technology.

- Margolis, Michael (2011). Arduino Cookbook. O'Reilly Media, Incorporated
- Matus Lerner M. (2019). Latin American NIMEs: Electronic Musical Instruments and Experimental Sound Devices in the Twentieth Century
- McVay, J., Kapur A. (2012). MechBass: A Systems Overview of a New Four-Stringed Robotic Bass Guitar. Carnegie School of Engineering
- Murphy Jim, Carnegie Dale A., Kapur Ajay (2014). Little Drummer Bot: Building, Testing, and Interfacing With a New Expressive Mechatronic Drum System.
- Murphy Jim, Kapur Ajay, Carnegie Dale A (2012). Interacting with solenoid drummers: a quantitative approach to composing and performing with open-loop solenoid-based robotic percussion systems.
- Murphy Jim, Kapur Ajay, Carnegie Dale A (2014). Better Drumming Through Calibration: Techniques for Pre-Performance Robotic Percussion Optimization. NIME'12, May 21 – 23, 2012, University of Michigan, Ann Arbor.
- ONU (2019). A New Circular Vision for Electronics. Time for a Global Reboot. Para el World Economic Forum de 2019.
- Peñaloza B., Narvaez C. Solanes F. (2014). Residuos de Aparatos Eléctricos y Electrónicos: Su problemática en Argentina. Para el 14º Simposio Argentino de Informática y Derecho, SID 2014.
- Raes, Godfried-Willem (1993). A Personal story of Music and Technologies. Ghent University College - School of Arts & Logos Foundation.
- Raes, Godfried-Willem (1987-2020). Expression control in automated musical instruments; an ongoing survey and report. Ghent University College - School of Arts & Logos Foundation.
- Raes, Godfried-Willem. Catalogue of instruments and automats (Chronological). https://www.logosfoundation.org/godfried/instrum-god.html
- Reid, S., Sithi-Amnua, S., Kapur, A. (2018). Women Who Build Things: Gestural Controllers, Augmented Instruments, and Musical Mechatronics. NIME'18, June 3-6, 2018, Blacksburg, Virginia, USA.
- Rogers T., Kemper S. y Barton S. (2015). MARIE: Monochord-Aerophone Robotic Instrument Ensemble. Proceedings of the 2015 International Conference on New Interfaces for Musical Expression (NIME´15). Baton Rouge, LA: Louisiana State University.
- Singer E., Feddersen J., Redmon C., Bowen B. (2004). LEMUR's Musical Robots. En Proceedings of the 2004 Conference on New Interfaces for Musical Expression (NIME04), Hamamatsu, Japan

Anexo

1. Código Fuente

1.1. Main.cpp

```
/*
 *
 *  _____           _                  /\
 *  |  __ \         | |                /  \
 *  | |__) |  _   _ | | ___          / /\ \    _ __
 *  |  ___/  | | | | | |/ __|       / /\ \   | '__|
 *  | |      | |_| | | |\__ \ _    / ____ \  | |
 *  |_|       \__,_| |_||___/ (_) /_/    \_\ |_|
 *
 *  Sistema mecatrónico de percusión
 *  construido con materiales recuperados
 *
 *  Tesis de grado de Nicolás Miñán
 *
 *  v0.3
 */

#include <Arduino.h>
#include <MIDI.h>
#include <vector>
#include "PulsArControlador.h"

/*****************
 * Configuración *
 *****************/

boolean desarrollo = false;  // Control de Desarrollo

// Configuracción de Solenoides Micro 1
std::vector<std::array<int, 2>> PulsArSolenoides_micro1 = {
        /* Nota    PinOut */
/* 1 */ {   60,    13   },
/* 2 */ {   61,    12   },
/* 3 */ {   62,    14   },
/* 4 */ {   63,    27   },
/* 5 */ {   64,    26   },
/* 6 */ {   65,    25   },
/* 7 */ {   66,    33   },
/* 8 */ {   67,    32   },
};

// Configuracción de PAPs Micro 1
std::vector<std::array<int, 7>> PulsArAgitPAP_micro1 = {
       /* NotaSH  NotaAH NotaAct PrGiroCC PasoPO  DirPO   ActPO */
/* 1 */ {   60,     61,     62,     12,     2,      15,      0   },
/* 2 */ {   63,     64,     65,     13,     5,      4,      18  },
/* 2 */ {   66,     67,     68,     14,     21,     19,     22   },

};

// Configuracción de Servos Micro 1
std::vector<std::array<int, 4>> PulsArServos_micro1 = {};



// Configuracción de Solenoides Micro 2
```

```
std::vector<std::array<int, 2>> PulsArSolenoides_micro2 = {
    /* Nota    PinOut */
/* 1 */ {   60,    13   },
/* 2 */ {   61,    12   },
/* 3 */ {   62,    14   },
/* 4 */ {   63,    27   },
/* 5 */ {   64,    26   },
/* 6 */ {   65,    33   },
/* 7 */ {   66,    32   },
/* 7 */ {   67,    0   },
};

// Configuracción de PAPs Micro 2
std::vector<std::array<int, 7>> PulsArAgitPAP_micro2 = {};

// Configuracción de Servos Micro 2
std::vector<std::array<int, 4>> PulsArServos_micro2 = {
     /*  PosCC PosMinCC PosMaxCC  PinOut */
/* 1 */ {   12,     13,     14,     15  }, // Tambor Agudo Parche
/* 2 */ {   15,     16,     17,     2   }, // Tambor Agudo Percutor En Descanso
/* 3 */ {   18,     19,     20,     4   }, // Tambor Grave Parche
/* 4 */ {   21,     22,     23,     5   }, // Tambor Grave Percutor En Descanso
/* 5 */ {   24,     25,     26,     18  },
/* 6 */ {   27,     28,     29,     19  },
/* 7 */ {   30,     31,     32,     21  },
};


/*****************
 * Kernel        *
 *****************/

 /** Variables Operativas **/

boolean sistema_iniciado = false;
midi::MidiInterface<HardwareSerial>                     InterfazMIDI                     =
midi::MidiInterface<HardwareSerial>(Serial1); // Interfaz MIDI


//   PulsArControlador   PulsArControlador(PulsArSolenoides_micro1,   PulsArAgitPAP_micro1,
PulsArServos_micro1); //  PulsArControlador 1

PulsArControlador      PulsArControlador(PulsArSolenoides_micro2,      PulsArAgitPAP_micro2,
PulsArServos_micro2); //  PulsArControlador 2



/** Funciones Operativas **/

void iniciarSistema(){
    if(sistema_iniciado == false){
        delay(2000);
        if(desarrollo == true){
            Serial.println("");
            Serial.println("  _____                _");
            Serial.println(" |  __ \\              | |              /\\");
            Serial.println(" | |__) |  _   _     | |     ___     /  \\      _ __");
            Serial.println(" |  ___/  | | | |    | |    / __|   / /\\ \\    | '__|");
            Serial.println(" | |      | |_| |    | |    \\__ \\  / ____ \\   | |");
            Serial.println(" |_|       \\__,_|   |_|    |___/ (_)/_/    \\_\\ |_|");
            Serial.println("");
            Serial.println("PulsAr v0.3.200913");
        }
        sistema_iniciado = true;
    }
}
void _NoteOn_cb(byte _canal, byte _nota, byte _velocidad){
    PulsArControlador.procesarNoteOnOff(_canal, _nota, _velocidad);
}
void _NoteOff_cb(byte _canal, byte _nota, byte _velocidad){
    PulsArControlador.procesarNoteOnOff(_canal, _nota, 0);
}
```

```
void _CC_cb(byte _canal, byte _ccn, byte _valor){
    PulsArControlador.procesarCC(_canal, _ccn, _valor);
}


/************************
 * SETUP *
 ************************/

void setup() {
    btStop();
    /* Setup Interfaz MIDI */
    InterfazMIDI.begin(MIDI_CHANNEL_OMNI);  //Inicializo la interfaz
    InterfazMIDI.turnThruOff();
    InterfazMIDI.setHandleNoteOn(_NoteOn_cb); //Establezco Callback para NoteOn
    InterfazMIDI.setHandleNoteOff(_NoteOff_cb); //Establezco Callback para NoteOn
    InterfazMIDI.setHandleControlChange(_CC_cb); // Establezco Callback para CC

    PulsArControlador.inicializar();
                                                    /* cSl cPP cSm  */
    // /* MICRO 1 */ PulsArControlador.configurarCanales(  1,  2,  3   );
 /* MICRO 2 */ PulsArControlador.configurarCanales(  4,  2,  3   );
    PulsArControlador.configurarMonitorMIDI(23, 5000);

    if(desarrollo == true){
        // Monitor: minicom -D /dev/ttyUSB0 -b 9600
        Serial.begin(500000); // Inicio Puerto Serie 0 para monitoreo
    }

}
/************************
 * LOOP *
 ************************/
void loop() {
    InterfazMIDI.read();
    PulsArControlador.ejecutarCentinela();
}
```

1.2. PulsArControlador.h

```
#include <vector>
#include <map>
#include <string>
#include <Arduino.h>

#include "PulsArSolenoide.h"
#include "PulsArServo.h"
#include "PulsArAgitPAP.h"

extern boolean desarrollo;

class PulsArControlador{
    private:
        std::map<std::string, boolean> debug = {
            {"procesarNoteOnOff", false},
            {"procesarNoteOnOff_NR", false},
            {"procesarCC", false},
            {"procesarCC_NR", false},
        };

        // Vector de PulsArSolenoides
        std::vector<PulsArSolenoide> PulsArSolenoides;
        unsigned char canalPulsArSolenoides = 1;
        std::map<int, int> mapaPulsArSolenoidesNota;

        std::vector<PulsArAgitPAP> PulsArAgitPAPs;
        unsigned char canalPulsArAgitPAPs = 1;
        std::map<int, int> mapaPulsArAgitPAPsNotaSH;
        std::map<int, int> mapaPulsArAgitPAPsNotaAH;
```

```
        std::map<int, int> mapaPulsArAgitPAPsNotaAct;
        std::map<int, int> mapaPulsArAgitPAPsCCProfGiro;

        std::vector<PulsArServo> PulsArServos;
        unsigned char canalPulsArServos = 1;
        std::map<int, int> mapaPulsArServosPos;
        std::map<int, int> mapaPulsArServosPosMin;
        std::map<int, int> mapaPulsArServosPosMax;

        unsigned long ultInicioEventoMIDI = 0;
        unsigned int durMonitorMIDI = 50 * 1000; // en uS
        unsigned int monitorMIDIPinSalida = 23;

    public:

        PulsArControlador( // Constructor

            std::vector<std::array<int, 2>> _PulsArSolenoides,
            std::vector<std::array<int, 7>> _PulsArAgitPAPs,
            std::vector<std::array<int, 4>> _PulsArServos
        ){
            this->instanciarPulsArSolenoides(_PulsArSolenoides);
            this->instanciarPulsArAgitPAPs(_PulsArAgitPAPs);
            this->instanciarPulsArServos(_PulsArServos);
        }

        void inicializar(){
            for(int i = 0; i < this->PulsArServos.size(); i++){
                this->PulsArServos[i].inicializar();
            }
        }

        void configurarCanales(
            unsigned char _canalPulsArSolenoides,
            unsigned char _canalPulsArAgitPAPs,
            unsigned char _canalPulsArServos
        ){
            this->canalPulsArSolenoides = _canalPulsArSolenoides;
            this->canalPulsArAgitPAPs = _canalPulsArAgitPAPs;
            this->canalPulsArServos = _canalPulsArServos;
        }

        void configurarMonitorMIDI(
            unsigned int _monitorMIDIPinSalida,
            unsigned int _durMonitorMIDI
        ){
            this->durMonitorMIDI = _durMonitorMIDI;
            this->monitorMIDIPinSalida = _monitorMIDIPinSalida;

            pinMode(this->monitorMIDIPinSalida, OUTPUT);
        }

        void procesarNoteOnOff(byte _canal, byte _nota, byte _velocidad){
            if (
                _canal == this->canalPulsArSolenoides
                && this->mapaPulsArSolenoidesNota.count(_nota) > 0
            ){
                                                                this->PulsArSolenoides[this-
>mapaPulsArSolenoidesNota[_nota]].procesarNoteOnOff(_velocidad);
            }else if(
                _canal == this->canalPulsArAgitPAPs
                && this->mapaPulsArAgitPAPsNotaSH.count(_nota) > 0
            ){
                                                                  this->PulsArAgitPAPs[this-
>mapaPulsArAgitPAPsNotaSH[_nota]].procesarNoteOn(0, _velocidad);
            }else if(
                _canal == this->canalPulsArAgitPAPs
                && this->mapaPulsArAgitPAPsNotaAH.count(_nota) > 0
            ){
                                                                  this->PulsArAgitPAPs[this-
>mapaPulsArAgitPAPsNotaAH[_nota]].procesarNoteOn(1, _velocidad);
            }else if(
```

```
                _canal == this->canalPulsArAgitPAPs
                && this->mapaPulsArAgitPAPsNotaAct.count(_nota) > 0
            ){
                                                                this->PulsArAgitPAPs[this-
>mapaPulsArAgitPAPsNotaAct[_nota]].procesarNoteOn(2, _velocidad);
            }else{
                if(desarrollo == true && this->debug["procesarNoteOnOff_NR"] == true){
                    Serial.print("[CTRL]    ");
                    Serial.print("##Nota N/R## >> Canal: ");
                    Serial.print(_canal);
                    Serial.print(" | Nota: ");
                    Serial.print(_nota);
                    Serial.print(" | Vel: ");
                    Serial.println(_velocidad);
                }
            }

            this->registrarMonitorMIDI();

            if(desarrollo == true && this->debug["procesarNoteOnOff"] == true){
                Serial.print("[CTRL]    ");
                Serial.print("procesarNoteOnOff >> Canal: ");
                Serial.print(_canal);
                Serial.print(" | Nota: ");
                Serial.print(_nota);
                Serial.print(" | Velocidad: ");
                Serial.println(_velocidad);
            }
        }

        void procesarCC(byte _canal, byte _ccn, byte _valor){
             if (_canal == this->canalPulsArServos && this->mapaPulsArServosPos.count(_ccn) >
0 ) {
                                                                this->PulsArServos[this-
>mapaPulsArServosPos[_ccn]].procesarCCPosicion(_valor);
                                }else   if(_canal   ==   this->canalPulsArServos   &&   this-
>mapaPulsArServosPosMin.count(_ccn) > 0 ){
                                                                this->PulsArServos[this-
>mapaPulsArServosPosMin[_ccn]].procesarCCPosicionMin(_valor);
                                }else   if(_canal   ==   this->canalPulsArServos   &&   this-
>mapaPulsArServosPosMax.count(_ccn) > 0 ){
                                                                this->PulsArServos[this-
>mapaPulsArServosPosMax[_ccn]].procesarCCPosicionMax(_valor);
                               }else   if(_canal   ==   this->canalPulsArAgitPAPs   &&   this-
>mapaPulsArAgitPAPsCCProfGiro.count(_ccn) > 0 ){
                                                                this->PulsArAgitPAPs[this-
>mapaPulsArAgitPAPsCCProfGiro[_ccn]].procesarCCProfGiro(_valor);
            }else{
                if(desarrollo == true && this->debug["procesarCC_NR"] == true){
                    Serial.print("[CTRL]    ");
                    Serial.print("##CC N/R## >> Canal: ");
                    Serial.print(_canal);
                    Serial.print(" | Nota: ");
                    Serial.print(_ccn);
                    Serial.print(" | Valor: ");
                    Serial.println(_valor);
                }
            }

            this->registrarMonitorMIDI();

            if(desarrollo == true && this->debug["procesarCC"] == true){
                Serial.print("[CTRL]    ");
                Serial.print("procesarCC >> Canal: ");
                Serial.print(_canal);
                Serial.print(" | CCn: ");
                Serial.print(_ccn);
                Serial.print(" | Valor: ");
                Serial.println(_valor);
            }
        }
```

```
        void ejecutarCentinela(){
            for(int i = 0; i < this->PulsArSolenoides.size(); i++){
                this->PulsArSolenoides[i].ejecutarCentinela();
            }

            // for(int i = 0; i < this->PulsArServos.size(); i++){
            //     this->PulsArServos[i].ejecutarCentinela();
            // }

            for(int i = 0; i < this->PulsArAgitPAPs.size(); i++){
                this->PulsArAgitPAPs[i].ejecutarCentinela();
            }

            this->actualizarMonitorMIDI();
        }

    private:
        void instanciarPulsArSolenoides(std::vector<std::array<int, 2>>  _PulsArSolenoides){
            for(int i = 0; i < _PulsArSolenoides.size(); i++){

                //Mapeo el id del solenoide en mapaPulsArSolenoidesNota
                this->mapaPulsArSolenoidesNota[_PulsArSolenoides[i][0]] = i;

                //Instancio el PulsArSolenoide
                   this->PulsArSolenoides.push_back( PulsArSolenoide(i, _PulsArSolenoides[i]
[1]) );
            }
        }

        void instanciarPulsArAgitPAPs(std::vector<std::array<int, 7>> _PulsArAgitPAPs){
            for(int i = 0; i < _PulsArAgitPAPs.size(); i++){
                //Mapeo el id del servo en mapaServos
                this->mapaPulsArAgitPAPsNotaSH[_PulsArAgitPAPs[i][0]] = i;
                this->mapaPulsArAgitPAPsNotaAH[_PulsArAgitPAPs[i][1]] = i;
                this->mapaPulsArAgitPAPsNotaAct[_PulsArAgitPAPs[i][2]] = i;
                this->mapaPulsArAgitPAPsCCProfGiro[_PulsArAgitPAPs[i][3]] = i;

                //Instancio el PulsArServo
                this->PulsArAgitPAPs.push_back( PulsArAgitPAP(
                    i,
                    _PulsArAgitPAPs[i][4],
                    _PulsArAgitPAPs[i][5],
                    _PulsArAgitPAPs[i][6]
                ));
            }
        }

        void instanciarPulsArServos(std::vector<std::array<int, 4>> _PulsArServos){
            for(int i = 0; i < _PulsArServos.size(); i++){

                //Mapeo el id del servo en mapaServos
                this->mapaPulsArServosPos[_PulsArServos[i][0]] = i;
                this->mapaPulsArServosPosMin[_PulsArServos[i][1]] = i;
                this->mapaPulsArServosPosMax[_PulsArServos[i][2]] = i;

                //Instancio el PulsArServo
                this->PulsArServos.push_back( PulsArServo(i, _PulsArServos[i][3]) );
            }
        }



        void registrarMonitorMIDI(){
            this->ultInicioEventoMIDI = micros();
        }
        void actualizarMonitorMIDI(){

            if(micros() - this->ultInicioEventoMIDI < this->durMonitorMIDI){
                digitalWrite(this->monitorMIDIPinSalida, HIGH);
                if(desarrollo == true && this->debug["actualizarMonitorMIDI"] == true){
                    Serial.println("ENCENDER MONMIDI");
                }
```

```
            }else{
                digitalWrite(this->monitorMIDIPinSalida, LOW);
                if(desarrollo == true && this->debug["actualizarMonitorMIDI"] == true){
                    Serial.println("APAGAR MONMIDI");
                }
            }


            if(desarrollo == true && this->debug["actualizarMonitorMIDI"] == true){
                Serial.print("micros(): ");
                Serial.print(micros());
                Serial.print(" | this->ultInicioEventoMIDI: ");
                Serial.print(this->ultInicioEventoMIDI);
                Serial.print(" | this->durMonitorMIDI: ");
                Serial.println(this->durMonitorMIDI);
            }
        }
};
```

1.3. PulsArSolenoide.h

```
#include <map>
#include <string>
#include <Arduino.h>

extern boolean desarrollo;

class PulsArSolenoide{

    private:
        std::map<std::string, boolean> debug = {
            {"procesarNoteOnOff", false},
        };

        // Configuración general de la clase
        unsigned int        tMaxActivo = 2000; // En milisegundos
                 float      tMinPausa = 0.8; // En milisegundos
        unsigned char       ledc_resBits = 7;
        unsigned int        ledc_frecuencia = 5000;

        // Propiedades de la instancia
                 char       id;
                 char       canalLED;
                 char       pinSalida;
        unsigned long       ultMomApagado = 0;
        unsigned long       ultMomEncendido = 0;
                 char       velocidad = 0;
                 char       velocidadComparador = 0;


    public:
        // Constructor
        PulsArSolenoide(int _id, int _pinSalida){
            // Registro información del solenoide
            this->id = _id;
            this->pinSalida = _pinSalida;

            // Realizo operaciones de inicialización del solenoide
            this->inicializar();
            // this->inicializarNoLedc();
        }

        void procesarNoteOnOff(int _velocidad){
            /*
            procesarNoteOnOff
            -----------------
            Registrar actualización de valores de KV
            */
            this->velocidad = _velocidad;
```

```
            if(desarrollo == true && this->debug["procesarNoteOnOff"] == true){
                Serial.print("[SOL]     ");
                Serial.print("procesarNoteOnOff >> #");
                Serial.print(this->id);
                Serial.print(" | Vel: ");
                Serial.println(_velocidad);
            }
        }

        void ejecutarCentinela(){
            /*
            ejecutarCentinela
            -----------------
            Ejecutar todos los procedimientos cíclicos del objeto
            */


            this->chequearActualizacionKV();

            // Chequeo si tengo que ejecutar parada de emergencia
            this->chequearTiempoDeActividad();
        }

    private:

        void inicializar(){
            /*
            inicializar
            -----------
            Inicializar puerto de salida
            */
            // ledcSetup(this->id, this->ledc_frecuencia, this->ledc_resBits);
            // ledcAttachPin(this->pinSalida, this->id);
            pinMode(this->pinSalida, OUTPUT);

            if(desarrollo == true){
                Serial.print("Inicializo PulsArSolenoide en canalLED ");
                Serial.print(this->id);
                Serial.print(" con pin salida ");
                Serial.println(this->pinSalida);
            }
        }

        void inicializarViejo(){
            /*
            inicializar
            -----------
            Inicializar puerto de salida
            */
            ledcSetup(this->id, this->ledc_frecuencia, this->ledc_resBits);
            ledcAttachPin(this->pinSalida, this->id);

            if(desarrollo == true){
                Serial.print("Inicializo PulsArSolenoide en canalLED ");
                Serial.print(this->id);
                Serial.print(" con pin salida ");
                Serial.println(this->pinSalida);
            }
        }

        void escribirActividadMotor(int _velocidad){
            /*
            escribirActividadMotor
            ----------------------
            Pp: - Escribir señal de control de motor.
                - Registrar momento (t) de inicio o fin de acción
            Pc: - Recibir por parámetro un int con el KV correspondiente
                - Que esté inicializado el puerto con su correspondiente canalLED
                - Que existan los campos, ultMomEncendido y ultMomApagado
            Obs: Se elige tomar el valor de ancho de pulso a partir de un
                parámetro del método para evitar que este funcione en cualquier
```

```
                circunstancia, con el KV registrado en la instancia.
            */
            // Escribo señal en puerto correspondiente
            // ledcWrite(this->id, _velocidad);
            if(_velocidad == 0){
                digitalWrite(this->pinSalida, LOW);
            }else{
                digitalWrite(this->pinSalida, HIGH);
            }


            // Registro el momento (t) del evento según corresponda
            if(_velocidad == 0){
                this->ultMomEncendido = micros();
            }else{
                this->ultMomApagado = micros();
            }
        }

        void escribirActividadMotorPWM(int _velocidad){
            /*
            escribirActividadMotor
            ----------------------
            Pp: - Escribir señal de control de motor.
                - Registrar momento (t) de inicio o fin de acción
            Pc: - Recibir por parámetro un int con el KV correspondiente
                - Que esté inicializado el puerto con su correspondiente canalLED
                - Que existan los campos, ultMomEncendido y ultMomApagado
            Obs: Se elige tomar el valor de ancho de pulso a partir de un
                parámetro del método para evitar que este funcione en cualquier
                circunstancia, con el KV registrado en la instancia.
            */
            // Escribo señal en puerto correspondiente
            ledcWrite(this->id, _velocidad);

            // Registro el momento (t) del evento según corresponda
            if(_velocidad == 0){
                this->ultMomEncendido = micros();
            }else{
                this->ultMomApagado = micros();
            }
        }

        void chequearTiempoDeActividad(){
            /*
            chequearTiempoDeActividad
            -------------------------
            Pp: Chequar si tengo que ejecutar apagado de seguridad
            */

            // Si está encendido...
            if(this->velocidad != 0){

                // Si se excedio el tiempo max de activo...
                if( micros() - (this->ultMomApagado) > this->tMaxActivo*1000 ){

                    // Apago el motor
                    this->escribirActividadMotor(0);

                }
            }
        }

        void chequearActualizacionKV(){
            /*
            chequearActualizacionKV
            -----------------------
               Pp: Ejecutar los procedimientos requeridos, en caso de actualización de KV.
Estos son:
                - Escritura actividad motor
                - Actualización de comparador de KV
            */
```

```
            // Si hay un nuevo KV...
            if(this->velocidadComparador != this->velocidad){

                // Si estoy encendiendo el motor (el comparador estaba en 0)...
                if(this->velocidadComparador == 0){

                    // Si se cumplió la pausa reglamentaria...
                    if( micros() - (this->ultMomEncendido) > this->tMinPausa*1000 ){

                        // Escribo actividad el motor
                        this->escribirActividadMotor(this->velocidad);

                    }

                }
                // Sino, si estoy apagando el motor
                else if(this->velocidad == 0){
                    this->escribirActividadMotor(this->velocidad);
                }

                // Actualizo comparador
                this->velocidadComparador = this->velocidad;
            }
        }
};
```

1.4. PulsArAgitPAP.h

```
#include <map>
#include <string>
#include <Arduino.h>
#include <AccelStepper.h>

extern boolean desarrollo;

class PulsArAgitPAP{

    private:
        std::map<std::string, boolean> debug = {
            {"procesarNoteOn", true},
            {"procesarCCProfGiro", true},
            {"convertirCCMIDIARangoPasos", false},
            {"cambiarEstado", true},
            {"pedirGiroPAP", false},
            {"moverPAP", false},
        };

        // Configuración general de la clase
                int        profGiroMin = 1; // 48/4
                 int        profGiroMax = 40; // 48*8 para completar un giro en 1/8 de paso
micropaso

        // Propiedades de la instancia
        unsigned char       id;
                 char       controlPasoPinSalida;
                 char       controlDireccionPinSalida;
                 char       controlActivacionPinSalida;

                 boolean    estado = false;
                 int        profundidadGiro = 48; // Pasos
                 boolean    direccionGiro = true;

        AccelStepper        stepper;

        unsigned long       ultMomGiro = 0;
        unsigned int        maxTiempoInact = 5 * 1000 * 1000;

    public:
```

```
    // Constructor
    PulsArAgitPAP(
        int _id,
        int _controlPasoPinSalida,
        int _controlDireccionPinSalida,
        int _controlActivacionPinSalida
    ){
        // Registro información del solenoide
        this->id = _id;
        this->controlPasoPinSalida = _controlPasoPinSalida;
        this->controlDireccionPinSalida = _controlDireccionPinSalida;
        this->controlActivacionPinSalida = _controlActivacionPinSalida;

        this->stepper = AccelStepper(
            AccelStepper::DRIVER,
            this->controlPasoPinSalida,
            this->controlDireccionPinSalida
        );
        this->stepper.setAcceleration(50000);
        this->stepper.setEnablePin(this->controlActivacionPinSalida);
        this->stepper.setPinsInverted (false, false, true);
    }

    void procesarNoteOn(int _tipo, int _velocidad){
        /*
        procesarNoteOn
        ----------------

        */
        if(_tipo != 2){
            this->pedirGiroPAP(_tipo, _velocidad);
        }else{
            this->cambiarEstado(_velocidad);
        }

        if(desarrollo == true && this->debug["procesarNoteOn"] == true){
            Serial.print("[PAP]     ");
            Serial.print("procesarNoteOn >> #");
            Serial.print(this->id);
            Serial.print(" | dir: ");
            if(_tipo == 1){
                Serial.print("SH");
            }else if(_tipo == 0){
                Serial.print("SA");
            }else if(_tipo == 2){
                Serial.print("ACT/DES");
            }
            Serial.print(" | vel: ");
            Serial.println(_velocidad);
        }
    }

    void procesarCCProfGiro(int _CCProfGiro){
        this->setearDatoProfGiro(_CCProfGiro);

        if(desarrollo == true && this->debug["procesarCCProfGiro"] == true){
            Serial.print("[PAP]     ");
            Serial.print("procesarCCProfGiro >> #");
            Serial.print(this->id);
            Serial.print(" | _CCProfGiro: ");
            Serial.println(_CCProfGiro);
        }
    }

    void ejecutarCentinela(){
        this->stepper.run();

        // Chequeo si tengo que ejecutar parada detención por inactividad
        this->chequearTiempoDeInactividad();
    }
private:
    void pedirGiroPAP(int _direccion, int _velocidad){
```

```
    if(_velocidad > 0){
        if(this->estado == false){
            this->encenderDriver();
        }
        this->registrarMomentoGiroPAP();
        this->setearVelocidadPAP(_velocidad);
        this->setearDireccionPAP(_direccion);
        this->moverPAP();
    }

    if(desarrollo == true && this->debug["pedirGiroPAP"] == true){
        Serial.print("pedirGiroPAP >> #");
        Serial.print(this->id);
        Serial.print(" | dir: ");
        if(_direccion == 1){
            Serial.print("SH");
        }else if(_direccion == 0){
            Serial.print("SA");
        }
        Serial.print(" | vel: ");
        Serial.println(_velocidad);
    }
}

void registrarMomentoGiroPAP(){
    this->ultMomGiro = micros();
}

void setearVelocidadPAP(int _velocidad){
    float velocidadMotor = (float) _velocidad / 127.0 * 5000.00;
    this->stepper.setMaxSpeed(velocidadMotor);


    if(desarrollo == true && this->debug["velocidadMotor"] == true){
        Serial.print("velocidadMotor >> #");
        Serial.print(this->id);
        Serial.print(" | velocidadMotor: ");
        Serial.println(velocidadMotor);
    }

}

void setearDireccionPAP(int _direccion){
    if(_direccion == 1){
        this->direccionGiro = true;
    }else{
        this->direccionGiro = false;
    }
}

void moverPAP(){
    this->stepper.stop();
    int factor = 1;
    if(this->direccionGiro == false){
        factor = -1;
    }
    this->stepper.move(factor * this->profundidadGiro);

    if(desarrollo == true && this->debug["moverPAP"] == true){
        Serial.print("moverPAP >> #");
        Serial.print(this->id);
        Serial.print(" | targetPosition: ");
        Serial.print(this->stepper.targetPosition());
        Serial.print(" | maxSpeed: ");
        Serial.print(this->stepper.maxSpeed());
        Serial.print(" | currentPosition: ");
        Serial.println(this->stepper.currentPosition());
    }
}

void setearDatoProfGiro(int _CCProfGiro){
    /*
```

```
            setearDatoProfGiro
            -----------------
            Pp: establece el dato de profundidad de giro (cantidad de pasos para
            completar un pedido de giro), entre:
                Mín: 5 (redondeo de 48/10)
                Máx: 384 (48 * 8) para completar un giro en 1/8 de paso micropaso
            */
            this->profundidadGiro = this->convertirCCMIDIARangoPasos(_CCProfGiro);
        }

        void cambiarEstado(int _velocidad){
            if(_velocidad != 0 && _velocidad < 64){
                this->apagarDriver();
            }else{
                this->encenderDriver();
            }
            if(desarrollo == true && this->debug["cambiarEstado"] == true){
                Serial.print("cambiarEstado >> #");
                Serial.print(this->id);
                Serial.print(" | estado: ");
                if(this->estado == false){
                    Serial.print("APAGADO");
                }else if(this->estado == true){
                    Serial.print("ENCENDIDO");
                }
                Serial.print(" | vel: ");
                Serial.println(_velocidad);
            }
        }

        void chequearTiempoDeInactividad(){
            if(this->estado == true){
                if(micros() - this->ultMomGiro > this->maxTiempoInact){
                    this->apagarDriver();
                }
            }
        }

        void apagarDriver(){
            this->estado = false;
            this->stepper.disableOutputs();
        }

        void encenderDriver(){
            this->estado = true;
            this->stepper.enableOutputs();
        }

        int convertirCCMIDIARangoPasos(int _CCProfGiro){
            float rangoGiro = profGiroMax - profGiroMin;
            float pasos = (float)_CCProfGiro / 127.0 * rangoGiro + profGiroMin;

            if(desarrollo == true && this->debug["convertirCCMIDIARangoPasos"] == true){
                Serial.print("convertirCCMIDIARangoPasos >> #");
                Serial.print(this->id);
                Serial.print(" | _CCProfGiro: ");
                Serial.print(_CCProfGiro);
                Serial.print(" | pasos: ");
                Serial.println(pasos);
            }

            return (int)pasos;
        }
};
```

1.5. PulsArServo.h

```
#include <map>
#include <string>
```

```
#include <Arduino.h>
#include <ESP32Servo.h>


extern unsigned long uSeg;
extern boolean desarrollo;

class PulsArServo{

    private:
        std::map<std::string, boolean> debug = {
            {"procesarCCPosicion", false},
            {"procesarCCPosicionMin", false},
            {"procesarCCPosicionMax", false},
            {"escribirPosicionServo", false},
        };

        Servo Servomotor;

        // Configuración general de la clase

        // Propiedades de la instancia
                 int        id;
                 int        pinSalida;
                 float      posicion = .5;
                 float      posicionMin = .4;
                 float      posicionMax = .6;

    public:
        // Constructor
        PulsArServo(int _id, int _pinSalida){
            // Registro información del solenoide
            this->id = _id;
            this->pinSalida = _pinSalida;

        }

        void inicializar(){
            /*
            inicializar
            -----------
            Inicializar el PulsArServo
            */
            this->Servomotor.attach(this->pinSalida);
        }

        void procesarCCPosicion(int _CCPos){
            /*
            procesarNoteOnOff
            -----------------
            Convertir CCPos (dato midi 0-127) en un valor normalizado (0.0-1.0)
            y usarlo para setear el dato de posición.
            */

            this->setearDatoPosicion( _CCPos / 127.0 );

            if(desarrollo == true && this->debug["procesarCCPosicion"] == true){
                Serial.print("[SRV]     ");
                Serial.print("procesarCCPosicion | Servo #");
                Serial.print(this->id);
                Serial.print(" | CC Pos: ");
                Serial.println(_CCPos);
            }
        }

        void procesarCCPosicionMin(int _CCPosMin){
            /*
            procesarCCPosicionMin
            ---------------------
            */

            this->setearDatoPosicionMin( _CCPosMin / 127.0 );
```

```
            if(desarrollo == true && this->debug["procesarCCPosicionMin"] == true){
                Serial.print("[SRV]     ");
                Serial.print("procesarCCPosicionMin | Servo #");
                Serial.print(this->id);
                Serial.print(" | CC Pos Min: ");
                Serial.println(_CCPosMin);
            }
        }
        void procesarCCPosicionMax(int _CCPosMax){
            /*
            procesarCCPosicionMax
            ---------------------
            */
            this->setearDatoPosicionMax( _CCPosMax / 127.0 );

            if(desarrollo == true && this->debug["procesarCCPosicionMax"] == true){
                Serial.print("[SRV]     ");
                Serial.print("procesarCCPosicionMax | Servo #");
                Serial.print(this->id);
                Serial.print(" | CC Pos Max: ");
                Serial.println(_CCPosMax);
            }
        }


    private:


        void setearDatoPosicion(float _pos){
            this->posicion = _pos;
            this->escribirPosicionServo();
        }
        void setearDatoPosicionMin(float _pos){
            this->posicionMin = _pos;
            this->escribirPosicionServo();
        }
        void setearDatoPosicionMax(float _pos){
            this->posicionMax = _pos;
            this->escribirPosicionServo();
        }
        void escribirPosicionServo(){
            /*
            escribirPosicionServo
            ---------------------
               |>> Toma los valores de posición, posición min y posición max y calcula la
posición en grados del servo, y la escribe.
            */
                  float posicionServoAEscribir = this->calcularPosicionFinalEnGrados(this-
>posicion, this->posicionMin, this->posicionMax);
            this->Servomotor.write(posicionServoAEscribir);

            if(desarrollo == true && this->debug["escribirPosicionServo"] == true){
                Serial.print("[SRV]     ");
                Serial.print("escribirPosicionServo | Servo #");
                Serial.print(this->id);
                Serial.print(" | PinSalida: ");
                Serial.print(this->pinSalida);
                Serial.print(" | Pos: ");
                Serial.print(posicionServoAEscribir);
                Serial.println("*");
            }
        }

            float calcularPosicionFinalEnGrados(float _posicion, float _posicionMin, float
_posicionMax){
            float posicionFinalEnGrados =
                abs(
                    (
                        (
                            (_posicionMax - _posicionMin) * _posicion
                        )
```

```
                        + _posicionMin
                    )
                    * 180.0
                );
            return posicionFinalEnGrados;
        }
};
```