E.T.S. de Ingeniería Industrial,

Informática y de Telecomunicación

LevPet: una mascota levitante con computación afectiva

Grado en Ingeniería Informática

Trabajo Fin de Grado

Autora: Josune Sorbet Molina

Directoras: Amalia Ortiz y Sonia Elizondo

Pamplona, 6 Septiembre 2021

upna
Universidad Pública de Navarra
Nafarroako Unibertsitate Publikoa

CONTENIDO

AGRADECIMIENTOS

Me gustaría agradecer en primer lugar, la ayuda de los miembros de UpnaLab en todo lo que han podido y más. En concreto a: Josu Irisarri, por enseñarme a estañar; Íñigo Ezcurdia, por acompañarme y enseñarme a usar la cortadora; Sonia Elizondo, por interesarse a cada paso del proyecto y ofrecerme los consejos que necesitaba; Amalia Ortiz y Asier Marzo, por haber tenido una paciencia infinita cuando no era capaz de poner en marcha el proyecto y por animarme y confiar desde el principio hasta el final.

También querría dar las gracias a mi familia, en concreto a mis padres, por darme el empujón que me hacía falta y escucharme cuando lo necesitaba.

A todos los compañeros del laboratorio, pero en particular a Álvaro Lerga, por hacer de pasarme días enteros programando y usando *superglue* algo divertido.

Por último, a Xabier Jiménez, que me cuida y me apoya más de lo que nadie podría pedir.

Sin todos vosotros, no sé qué habría pasado con este trabajo.

RESUMEN

El interés por dotar a las máquinas de una inteligencia emocional con la que poder interactuar de forma sencilla y agradable tiene su origen en los años 90. En ese momento, comienzan los primeros estudios sobre computación afectiva y sale al mercado la primera mascota virtual.

Hoy en día, el mercado está repleto de este tipo de mascotas y este proyecto pretende realizar una con una particularidad única: la levitación.

Este trabajo consiste en el diseño, construcción y programación de un agente que levita magnéticamente. Se pretende que pueda realizar diversas tareas propias de una mascota, como jugar, recibir a las personas o reaccionar a las emociones percibidas.

La mascota robótica sería capaz de expresar emociones en relación con las detectadas en el usuario. Esto se reflejaría mediante los diferentes tipos de movimiento y de las distintas actividades. Además, podría realizar acciones que una mascota real no podría, como tocar un instrumento.

Esto se conseguirá con la unión del hardware y técnicas, entre otras, de reconocimiento facial y de seguimiento de objetos.

PALABRAS CLAVE

- Mascota virtual
- Mascota robótica
- Computación afectiva
- Levitación magnética
- Unity
- G-CODE

INTRODUCCIÓN

MOTIVACIÓN

La motivación principal fue la de querer realizar una mascota robótica con un toque distintivo. La levitación nos abría así las puertas a nuevas características para explorar. De esta manera, podíamos estudiar cómo poder aprovechar e incorporar la levitación en algo tan carismático como lo son las mascotas robóticas.

Este proyecto nos permitiría observar qué es lo que hace de un objeto inanimado algo con lo que encariñarse y, por el camino, aprender sobre cómo combinar la computación afectiva con algo tan mecánico como un levitador.

OBJETIVOS

El objetivo del proyecto es crear una mascota robótica que pueda levitar. Esto, lo podemos dividir en pequeñas marcas que pretendemos lograr:

- La creación de un agente que levite
- Conseguir que dicho agente pueda desplazarse
- Que pueda distinguir emociones
- Que sea capaz de interactuar con el usuario y con su entorno
- Que pueda expresar de manera creíble sus emociones
- Que realice acciones que corresponden a su estado de ánimo
- Que dichas acciones sean variadas y que estén en sintonía con las realizadas por una mascota virtual

TRABAJO RELACIONADO

LAS MASCOTAS VIRTUALES Y ROBÓTICAS

[1]En 1996 nació la muy conocida primera mascota virtual: El *tamagochi* [2]. Fue uno de los grandes fenómenos de los 90 por conseguir que millones de usuarios en todo el mundo lograran establecer un vínculo personal y sentimental con un ser virtual que no era más que un pequeño conjunto de píxeles.

Ilustración I Tamagotchi[3]

Esta historia empezó con Aki Maita, inventora y pedagoga japonesa. Ella estaba interesada en enseñar a los niños la responsabilidad de cuidar de una mascota. Para la mayoría de los jóvenes, una mascota no es más que un ser con el que jugar y divertirse e ignoran o dejan a los padres la responsabilidad de alimentarlos, limpiarlos y pasearlos todos los días. Con el *tamagochi* se pretendía introducir en los niños estas responsabilidades, y hacer de ellas algo entretenido y deseable.

Dos años después, en 1997, apareció en el mercado Furby, un peluche con cuerpo mecánico que podía mover los párpados, la boca, la cola, moverse y hablar. Cabe destacar que estas palabras las decía según el cuidado que había recibido, fomentando así el buen trato al juguete.

En 1999, llegó Aibo, el primer perro robot y en 2003 una foca robot diseñada para ayudar a ancianos con algunas enfermedades.

Como se puede observar, a parte de la intención de enseñar a los más jóvenes sobre responsabilidades, proveen entretenimiento y una forma de conectar emocionalmente con una máquina que resulta simpática e intrigante.

COMPUTACIÓN AFECTIVA

La Computación Afectiva[4], es la disciplina que estudia cómo crear máquinas que puedan reconocer, interpretar y responder apropiadamente a las emociones humanas.

Esta disciplina nace en los 90 y se consolida en 1997 con la publicación del primer libro sobre Computación Afectiva, escrito por la profesora Rosalind Picard (MIT). En un principio, los avances en esta disciplina fueron lentos, pero con el paso del tiempo y el avance de la tecnología, cada vez existen más equipos y laboratorios interesados en el estudio de la Computación Afectiva. Este campo de investigación involucra de manera principal a la informática, pero además necesita de personas expertas en el comportamiento humano, como psicólogos.

A grandes rasgos, un sistema de reconocimiento de emociones trata de captar, con sensores, señales relacionadas con nuestras emociones. Estos sensores podrían capturar: imagen, para interpretar gestos, dilatación de las pupilas o posición de la persona o entorno; audio, para detectar tono de la voz y palabras importantes; pulsaciones o tensión arterial, para medir el nivel de estrés o excitación por algo, y multitud de otras señales. Explicado así, parece algo demasiado artificial, pero nosotros también utilizamos este tipo de información para saber cuál es la emoción que está sintiendo otra persona. La gran diferencia es que los seres humanos llevamos miles de años de evolución o, en otras palabras, miles de años de entrenamiento para aprender a identificar de manera sencilla, rápida y fiable emociones humanas, tanto que realmente no somos conscientes de en qué nos fijamos para saber que alguien está bien o mal, simplemente “lo notamos”. Pero si fuésemos diseccionando en partes más pequeñas de qué forma sabemos que una persona está sintiéndose de determinada manera, lo mencionado sería un factor muy importante.

Para que una máquina aprenda a distinguir las emociones, se necesita una gran cantidad de datos sobre las señales que hemos decidido que son importantes para diferenciar una emoción de otra. Necesitamos muchos ejemplos de muchas personas experimentando emociones. Una vez tenemos toda esta información, tenemos que procesarla. En este paso entra en juego el Machine Learning, cuyo objetivo es obtener patrones en los datos que hemos recolectado que son informativos sobre el estado emocional de una persona.

Una vez la máquina ha aprendido los patrones que debe de tener en cuenta y con qué intensidad, cuando ahora queramos saber qué emoción está sintiendo una persona, le

pasaremos a nuestra máquina los datos de este nuevo sujeto (imagen, audio, pulsaciones, …) y esta la clasificará en una de las distintas emociones que tiene registradas según lo parecida que sea la información de la nueva persona con los patrones aprendidos de cada emoción.

Esto no significa que la Computación Afectiva solo sirva para detectar qué está sintiendo una persona ya que un sistema completo de Computación Afectiva involucraría también sistemas que permitan interpretar y generar emociones.

Las aplicaciones que podría tener son numerosas, desde la medicina (como monitorizar el dolor para entender y reducir el sufrimiento de enfermos o detectar cambios sutiles en personas que puedan ser indicativos de algún problema mental) hasta desarrollar compañeros artificiales para ancianos que viven en soledad.

A pesar de los numerosos laboratorios que existen dedicados a la investigación de la Computación Afectiva, aún no hay ningún sistema que reconozca y entienda a la perfección las emociones de una persona. Con respecto a los dispositivos que la mayoría de la población usa a diario (ordenadores, televisión, smartphone, …) no son emocionalmente inteligentes, no actúan de manera distinta si estamos tristes, preocupados o contentos, aunque nosotros sí consumamos distinto contenido en ellos en función de cómo nos sentimos.

Es por ello por lo que aún queda un largo camino hasta alcanzar el objetivo de la Computación Afectiva, pero los progresos conseguidos son prometedores y esperanzadores.

LEVITACIÓN MAGNÉTICA

<<Entendemos por levitación el efecto que produce la aplicación de una fuerza a un objeto, que cancela su peso permitiendo que se quede "flotando" en equilibrio. Quizá hayamos intentado hacer esto con dos imanes, buscando hacer levitar uno de ellos atrayéndolo con el otro situado por encima. Cualquier experiencia de este tipo seguro que habrá fallado, terminando el imán por caer o por subir y pegarse al otro. La justificación teórica de este hecho la dio **Earnshaw** en 1842. Su teorema indica, en resumen, que **es imposible mantener objetos cargados o imanes en equilibrio mediante fuerzas eléctricas, magnéticas o gravitatorias estáticas**. **Es posible encontrar puntos de equilibrio, pero son inestables.** La situación es análoga a cuando pretendemos mantener una varilla vertical apoyando su extremo inferior sobre una mesa. Sin embargo, si la apoyamos sobre la palma de nuestra mano y esta realiza los movimientos de control precisos, podremos mantener a la varilla en posición aproximadamente vertical. Es necesario que haya variaciones de las fuerzas aplicadas con el tiempo para conseguir el equilibrio. En forma análoga es posible suspender un objeto magnético mediante un electroimán del que se controla su corriente para poder ajustar la fuerza magnética sobre el objeto, disminuyéndola si se acerca demasiado al electroimán o aumentándola si se aleja demasiado.>>[6]

Para explicar de qué manera levita un imán [7] es necesario primero explicar que el problema no es que dos imanes no tengan la fuerza suficiente como para repelerse y que uno levite, el problema es que esto no es estable.

Cuando hablamos de estable nos referimos a que aunque el imán pueda estar levitando en equilibrio, en cuanto se produce un ligero desequilibrio, tiende a desestabilizarse aún más. Esto se manifiesta con el imán dándose la vuelta.

En un sistema estable, cuando se desestabiliza, se tiende a volver al estado de equilibrio.

La manera de solventar este problema es añadiendo electroimanes. Los electroimanes se apagarán y se encenderán para volver a poner el imán en un estado de equilibrio. Si nuestro imán empezara a inclinarse hacia un sentido, encenderíamos el electroimán correspondiente para repelerlo y que volviese a estar en el estado de equilibrio.

Lo único que se necesita es detectar si está desequilibrado y en qué sentido, y la velocidad suficiente para mandar a los electroimanes apagarse o encenderse en función de este estado.

Este tipo de sistema respeta el teorema de Earnshaw porque la cantidad de imanes encendidos no es siempre la misma para mantener al imán levitando, si no que están controlados para reaccionar a la desestabilidad encendiéndose o apagándose. Es decir, no se trata de fuerzas estáticas.

GCODE

<<G-code es un lenguaje de programación para máquinas CNC (control numérico por computadora). El código G significa "Código geométrico". Usamos este lenguaje para decirle a una máquina qué hacer o cómo hacer algo. Los comandos del código G indican a la máquina dónde moverse, qué tan rápido debe moverse y qué camino seguir.

Por ejemplo, en el caso de una máquina herramienta como un torno o un molino, la herramienta de corte es impulsada por estos comandos para seguir una trayectoria específica, cortando material para obtener la forma deseada.

De manera similar, en el caso de la fabricación aditiva o las impresoras 3D, los comandos del código G instruyen a la máquina para que deposite material, capa sobre capa, formando una forma geométrica precisa.>>[8]

En cuanto a los comandos[9], se dividen entre los '$' y los 'G'.

Los comandos '$' son los utilizados para modificar la configuración, ver o cambiar los estados y modos de ejecución de Grbl e iniciar un ciclo de inicio.

$$ (view Grbl settings)
$# (view # parameters)
$G (view parser state)
$I (view build info)
$N (view startup blocks)
$x=value (save Grbl setting)
$Nx=line (save startup block)
$C (check gcode mode)
$X (kill alarm lock)
$H (run homing cycle)
~ (cycle start)
! (feed hold)
? (current status)
ctrl-x (reset Grbl)

Los 4 últimos comandos se pueden utilizar cuando la máquina está en ejecución. Estos cambian inmediatamente el comportamiento de ejecución de Grbl o imprimen inmediatamente un informe de los datos importantes en tiempo real como la posición actual.

A continuación, vamos a destacar los más utilizados en el proyecto.

$$

Con este comando podemos ver la configuración actual del sistema. Todas estas configuraciones son persistentes y se mantienen en EEPROM, por lo que, si las apaga, se volverán a cargar la próxima vez que encienda su Arduino.

$0= (step pulse, usec)
$1= (step idle delay, msec)
$2= (step port invert mask:00000000)
$3= (dir port invert mask:00000110)
$4= (step enable invert, bool)
$5= (limit pins invert, bool)
$6= (probe pin invert, bool)
$10= (status report mask:00000011)
$11= (junction deviation, mm)
$12= (arc tolerance, mm)
$13= (report inches, bool)
$20= (soft limits, bool)
$21= (hard limits, bool)
$22= (homing cycle, bool)
$23= (homing dir invert mask:00000001)
$24= (homing feed, mm/min)
$25= (homing seek, mm/min)
$26= (homing debounce, msec)
$27= (homing pull-off, mm)
$100= (x, step/mm)
$101= (y, step/mm)
$102= (z, step/mm)
$110= (x max rate, mm/min)
$111= (y max rate, mm/min)
$112= (z max rate, mm/min)
$120= (x accel, mm/sec^2)
$121= (y accel, mm/sec^2)
$122= (z accel, mm/sec^2)
$130= (x max travel, mm)
$131= (y max travel, mm)

$132= (z max travel, mm)

$X

Este comando es utilizado para matar el estado de alarma de Grbl. Se entra en este estado cuando ha sucedido algo crítico, como llegar al final de carrera, un aborto durante un ciclo, introducir una posición que se sale de los límites establecidos, que desconozca su posición…

Si se entra en estado de alarma por desconocer la posición, se bloquearán todos los comandos hasta que no se haya realizado el '$H'.

$H

Este comando realiza el ciclo de inicio en Grbl. En otras palabras, lo utilizaremos para realizar homing al inicio de la ejecución y así conocer la posición.

?

Devuelve de manera inmediata el estado activo de Grbl y la posición actual en tiempo real, tanto en coordenadas de la máquina como en coordenadas de trabajo. En nuestro caso, la parte que nos interesa es la de saber si hay algún pin activo, es decir, saber si se ha llegado a final de carrera.

Los comandos 'G' son los utilizados para el movimiento, de esta forma mandamos la nueva posición destino, de qué manera queremos llegar a ella, a qué velocidad, el sistema de coordenadas (relativa/absoluta), …

Destacaremos algunos de los importantes para nuestro proyecto:

G1

Este comando indica a la máquina que se mueva en línea recta a una frame rate establecida. Especificamos la posición final con los valores X e Y, y la velocidad con el valor F

G17 / G18 / G18

Con estos comandos de código G seleccionamos el plano de trabajo de la máquina.

G17 - Plano XY
G18 - Plano XZ

G19 - Plano YZ

G90 / G91

Con los comandos G90 y G91 le decimos a la máquina qué sistema de coordenadas deseamos utilizar. G90 es para el modo absoluto y G91 es para el modo relativo.

En modo absoluto, no importa la posición anterior. El posicionamiento de la herramienta es siempre desde el punto absoluto/cero.

Por otro lado, en modo relativo, el posicionamiento de la herramienta es relativo al último punto. Entonces, si la máquina está actualmente en el punto (10,10), el comando G01 X10 Y5 llevará la herramienta al punto (20,15).

UNITY

Unity[10][11] nació en 2004 cuando tras fracasar el lanzamiento de su videojuego, David Helgason, Nicholas Francis y Joachim Ante decidieron utilizar las herramientas creadas durante el desarrollo de este.

A partir de este momento, el objetivo de este equipo sería el de crear un motor de videojuegos para pequeñas y grandes empresas. Querían, además, que programadores, artistas y diseñadores pudieran utilizar el entorno de manera interactiva y que no fuese necesario programar el juego específicamente para cada plataforma, si no que fuese apto para varias.

En un principio, tan solo serviría para Mac y había que pagar para usar el modo básico y el modo pro. Aunque funcionaba bien, no era aún el gigante que es ahora.

En 2008 y 2009 sucedieron dos acontecimientos esenciales para el auge de Unity. La compatibilidad con el iPhone en su lanzamiento, y la compatibilidad para Android. Además, la versión básica, pasaría a ser gratuita.

Por otro lado, a lo largo del tiempo, Unity ha demostrado ser una de las mejores opciones cuando hay necesidad de compartir un mismo sistema de trabajo, es decir, cuando dos personas pretenden cambiar un mismo proyecto sin que uno pise el trabajo del otro.

La versión actual es compatible con muchísimas plataformas (PC, Mac, Linux, iOS, Android, BlackBerry, PlayStation, Xbox, Wii, Wii U, Web…) y se puede programar tanto en C# como en C++.

Algunos ejemplos de juegos exitosos realizados por Unity son “Cuphead” o “Monument Valley”.

Ilustración II Juego Cuphead [12]

Las funcionalidades que tiene un motor de videojuegos como Unity incluye las siguientes:

- Motor gráfico para renderizar gráficos 2D y 3D
- Motor físico que simule las leyes de la física

- Animaciones
- Sonidos
- Inteligencia Artificial
- Programación o scripting

En resumen, Unity es una herramienta que te permite crear videojuegos para diversas plataformas (PC, videoconsolas, móviles, etc.) mediante un editor visual y programación vía scripting de manera sencilla e interactiva.

OPENCV

OpenCV [13] (Open Source Computer Vision Library) es una librería para visión por computadora y machine learning. Nació para proporcionar una infraestructura común para aplicaciones de visión por ordenador y al tener licencia BSD, facilita que empresas utilicen y modifiquen el código, lo cual hace de OpenCv una herramienta muy atractiva.

Esta biblioteca dispone de más de 2500 algoritmos de machine learning y visión por computadora. Alguno de los ejemplos para los que se pueden utilizar este tipo de algoritmos son los siguientes:

- Detección y reconocimiento de rostros
- Rastrear objetos en movimiento
- Reconocer paisajes
- Extraer modelos 3D

A día de hoy, esta librería tiene un número de descargas superior a 18 millones y más de 47 mil usuarios de comunidad. Este éxito además de en usuarios individuales, se ve reflejado en el uso del producto por grandes empresas (Google, Microsoft, Sony, Toyota, etc.), startups (Zeitera, Applied Minds, etc.), grupos de investigación e incluso organismos gubernamentales.

Dispone de interfaces C++, Python, Java y MATLAB y es compatible con Windows, Linux, Android y Mac OS. Está escrito en C++ y tiene una interfaz con plantilla que funciona con los contenedores STL.

En cuanto a la estructura [14] de OpenCV, se trata de una modular, esto es, que el paquete incluye varias bibliotecas compartidas o estáticas. Los módulos disponibles son los siguientes:

- Funcionalidad principal: un módulo que define las estructuras de datos básicas, incluida una matriz multidimensional y las funciones básicas utilizadas por los demás módulos.
- Procesamiento de imágenes (imgproc): un módulo de procesamiento de imágenes que incluye filtrado de imágenes tanto lineales como no lineales, transformaciones geométricas de imágenes, conversión de espacio de color, histogramas, etc.
- Análisis de video (video): un módulo de análisis de video que incluye algoritmos de estimación de movimiento, sustracción de fondo y seguimiento de objetos.

- Calibración de cámara y reconstrucción 3D (calib3d): algoritmos básicos de geometría de múltiples vistas, calibración de cámara única y estéreo, estimación de pose de objeto, algoritmos de correspondencia estéreo y elementos de reconstrucción 3D.
- Marco de características 2D (features2d): detectores de características destacadas, descriptores y comparadores de descriptores.
- Detección de objetos (objdetect): detección de objetos e instancias de las clases predefinidas
- GUI de alto nivel (highgui): una interfaz fácil de usar para funciones de IU simples.
- E / S de vídeo (videoio): una interfaz fácil de usar para captura de vídeo y códecs de video.

SISTEMA

DISEÑO DEL SISTEMA

DE LA SIMULACIÓN A LO TANGIBLE

A la hora de planificar el sistema, hemos pretendido en primer lugar, realizar una simulación 3D y a partir de ahí, avanzar con la parte tangible. La motivación por la cual se ha pretendido empezar por lo virtual, es que en este entorno se puede trabajar con situaciones ideales. En nuestro caso, podíamos así avanzar en el diseño ignorando las siguientes incertidumbres que nos planteaba el hardware:

- la altura del levitador
- la aceleración o velocidad máxima que podía soportar antes de caerse
- la necesidad de evitar el final del stage
- los problemas propios del gcode (como la imposibilidad de parar una instrucción sin tener que reiniciar el sistema, la necesidad de hacer homing antes de empezar para poder saber la posición, …)
- la necesidad de una estructura donde pudiera construirse el *playground* del agente

Para la simulación hemos utilizado Unity. Esta herramienta nos permite representar de manera sencilla los elementos finales del proyecto, sus movimientos y sus interacciones sin tener que preocuparnos de sus dificultades.

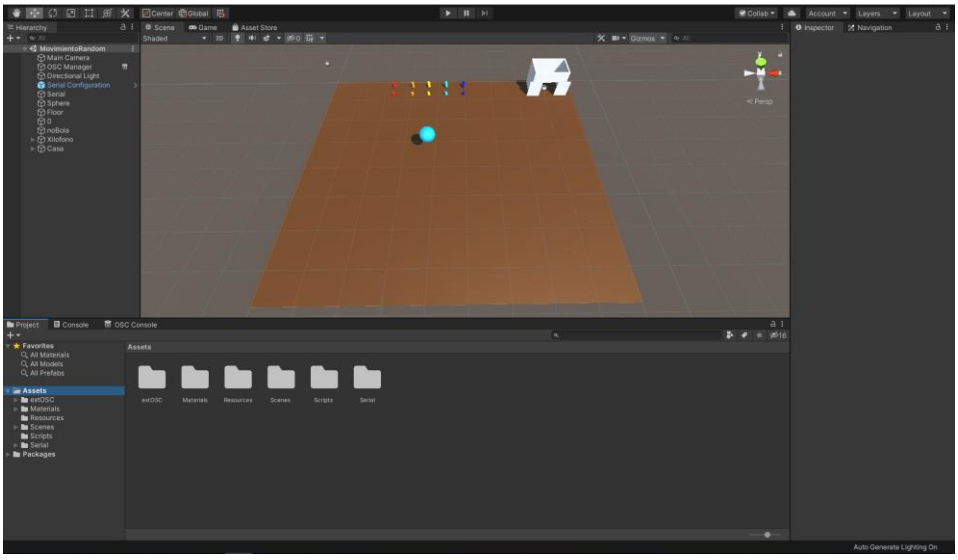

Ilustración III Simulación en Unity

Aquí, además de la mascota, hemos querido simular los elementos con los que va a interactuar: una pelota de ping-pong, un xilófono, su casa, la posición de la cara del usuario y su área de juego.

Ilustración IV LevPet real

VISTA GENERAL DEL SOFTWARE Y HARDWARE

DIAGRAMA

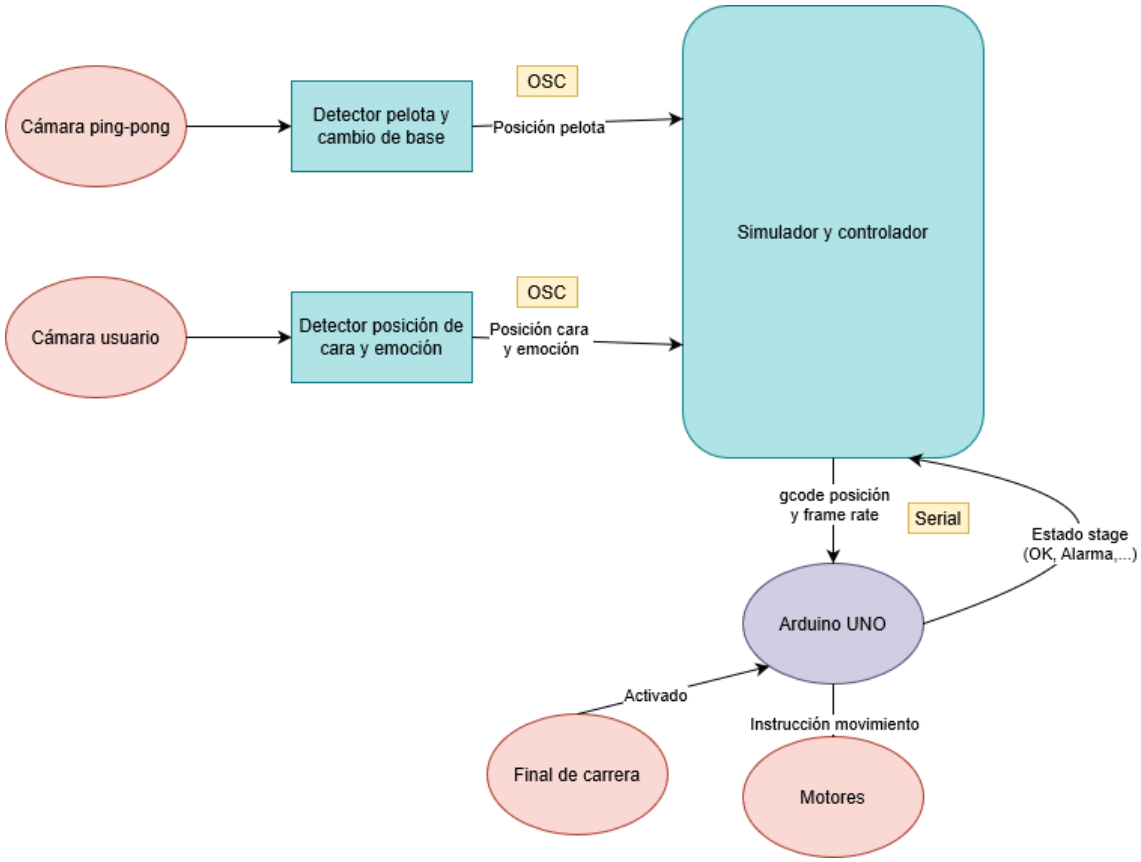

Ilustración V Diagrama componentes hardware y software

En este diagrama podemos observar las relaciones entre los distintos módulos y elementos que forman parte del sistema.

De color rojo, podemos ver la parte hardware. Por un lado, tenemos los inputs, que en este caso son las cámaras y el final de carrera. Por otro lado, tenemos los outputs, que son los motores del stage.

De color azul se representan los módulos de software. En primer lugar, tenemos el programa encargado de la detección del lugar de la pelota en el playground. En segundo lugar, el detector de la posición de la cara y la emoción manifestada. Por último, tenemos el simulador y controlador, que es el encargado de recibir información, procesarla y mandarla.

Un caso especial es el Arduino UNO, representado en morado, que, al ser un microcontrolador, es un elemento tanto hardware como software. Éste, es el punto intermedio entre el hardware de salida y el controlador.

En amarillo, están representadas las diferentes formas de comunicación entre el software. La utilizada para la relación entre los detectores y el controlador es el protocolo OSC. Por otro lado, entre el Arduino y el programa principal, se utiliza el Serial de manera bilateral.

BREVE DESCRIPCIÓN DE LAS INTERACCIONES CON EL USUARIO Y CON EL MEDIO

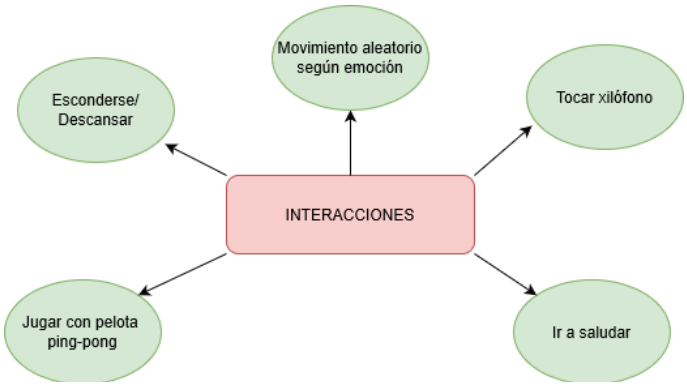

Ilustración VI Diagrama interacción con usuario y medio

Las distintas interacciones que tiene la mascota son las siguientes:

- Ir a saludar: Si la mascota lleva más de un tiempo determinado sin estar con un usuario, cuando este aparece, el agente irá a saludar. Es decir, se moverá hasta la posición de la persona, permanecerá ahí unos pequeños instantes y pasará a otro estado.
- Movimiento aleatorio según emoción: Cuando hay un usuario, se detecta la emoción que este está expresando y la mascota cambia su tipo de movimiento en función de la misma. Por ejemplo, si se detecta que la persona está contenta, la mascota también se alegrará y su movimiento será uno más energético. Por

otro lado, si el usuario estuviera enfadado, la mascota se asustaría y se movería más errática y precipitadamente.

- Esconderse/Descansar: La mascota dispone de una casa a la cual va en caso de estar asustado o de que lleve demasiado tiempo sin que haya ningún usuario cerca. Las razones por las cuales pueda estar asustada son que la persona esté asustada o que esté enfadada.
- Jugar con la pelota de ping-pong: Cuando en el *playground* se detecta esta pelota, la mascota irá a jugar con ella empujándola de un lado a otro del escenario.
- Tocar el xilófono: Si la mascota lleva un tiempo suficiente estando contenta, se irá a la parte del escenario donde está el xilófono y tocará las diferentes teclas para expresar alegría.

IMPLEMENTACIÓN DEL SISTEMA

HARDWARE

Estructural

Cuando hablamos de la parte estructural del hardware, nos referimos a lo relacionado con los componentes no eléctricos o electrónicos.

En nuestro caso, esto engloba los siguientes elementos:

- la construcción de la caja
- la construcción de la casa
- la construcción del xilófono
- método de colocación de la mascota si se ha caído

Caja

Ilustración VII Cortadora láser

Para la caja hemos usado una cortadora láser. En ella descargamos los planos de la caja [15] que queremos montar (previamente los hemos separado en archivos para cada tabla), ajustamos los parámetros de corte según el material y espesor de la madera y procedemos a cortar.

Realizamos varias versiones de cajas variando el grosor de los materiales, la altura, el ancho y el largo de la caja.

Para la primera versión de la caja, utilizamos madera de 3mm de espesor y unas medidas interiores de 73x75x23. A pesar de que no se combaba, resultó ser más pequeña de lo necesaria por el espacio ocupado por los motores.

Ilustración VIII Primera caja

Para realizar el diseño de la tapa de arriba, por no disponer de tablas lo suficientemente grandes, hemos usado AutoCad (software de diseño) para dibujar un corte en T y así poder encajar las dos mitades.

Para la siguiente versión de la caja usamos las siguientes medidas: grosor de 3mm y medidas interiores de 80x78x23 centímetros. En este caso el centro de la tapa quedaba hundido.

Es por esto que, para la versión final, cambiamos la tapa por otra de 5mm y mantuvimos el resto. Diseñamos la caja ligeramente más baja de lo necesario para posteriormente poder ajustarla a la altura óptima después de los ajustes que se necesitaran. Esto se consiguió añadiendo 3mm de elevación.

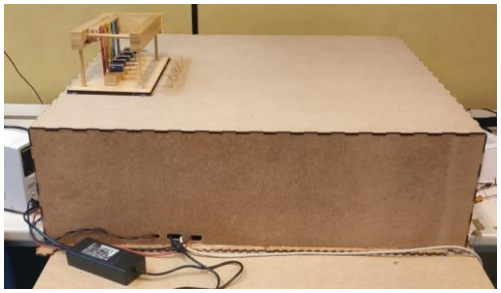

Ilustración IX Caja final

Casa

Ilustración X Casa de la mascota

Para la casa queríamos algo que pudiera destacar que la mascota no toca el suelo. Barajamos distintas posibilidades, pero la escogida fue una casa para pájaros. Para la construcción de la misma buscamos en un repositorio de distintas construcciones para la cortadora [16].

Una vez descargados los planos, modificamos la posición de la entrada con Inkscape (software de diseño) para que estuviera a una altura en la cual pudiese entrar, pero que no quedara totalmente a ras de suelo.

Xilófono

A la hora de elegir qué instrumento debía ser con el que interactuase la mascota, hubo que tener en cuenta los materiales por los que estuviera compuesto el mismo. En un principio, tratamos de excluir todos los materiales que pudieran afectar al levitador debido a sus propiedades magnéticas. Esto tan solo nos dejaba con instrumentos de madera o similares que no proyectaban de manera suficiente el sonido. Es por ello que

pensamos en otra forma de tocar instrumentos metálicos sin que ello implicase el contacto directo con ellos. Más adelante explicaremos de qué forma se consigue esto.

Ilustración XI teclado 25 notes glockespien

Tras decantarnos por un xilófono de metal, acotamos las teclas a utilizar a C (Do), D (Re), E (Mi), G (Sol) y A (La). Esto es una escala pentatónica [17] y es óptima para la improvisación y nuestro trabajo, ya que nos permite tocarlas en cualquier orden sin que nos parezca disonante.

Una vez seleccionadas las teclas, las colocamos en vertical en una estructura de madera para que esté en sintonía con el resto del playground.

Ilustración XII Xilófono en el proyecto

Método de colocación de la mascota

La colocación de la mascota en el punto exacto para la levitación no es sencilla. Por ello, desde un principio, diseñamos varias pequeñas estructuras con la intención de poder ayudar al usuario con esta tarea.

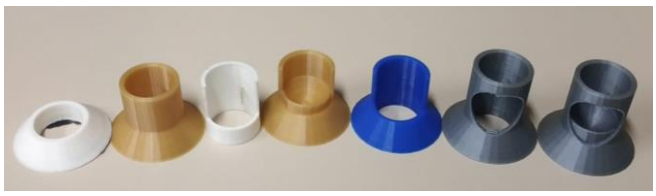

Ilustración XIII Ejemplos de estructuras utilizadas

Este tipo de figuras no ayudó con el problema, ya que nada más colocar a la mascota, esta caía atraída por el imán de la base del levitador.

Tras investigar diferentes tipos de soluciones[18], probamos a utilizar un tubo de aluminio para que la mascota, al ser un imán, cayera más despacio y que así la base del levitador pudiera soportar la caída. Esto es debido al tubo de Lenz[19], que explica cómo el imán que cae iría produciendo variaciones en el flujo del campo magnético dentro del tubo, dichas variaciones crean una fem (fuerza electromagnética) inducida que a su vez genera una corriente dentro del mismo. Como la fem y la corriente se oponen al cambio de flujo, se produce un nuevo campo que repele al imán por abajo y se atrae por arriba.

Componentes eléctricos y electrónicos

En este apartado vamos a hablar de los circuitos implementados y de los materiales utilizados.

Materiales

Solenoide 12V Efecto Lineal 3.7×1.5mm JF0530B [20]

Propiedades	**Descripción**
Nombre del producto	electroimán DC
Modelo	JF-0530B
Tensión nominal	DC 12 V
Tipo	push pull
Corriente nominal	300mA
Potencia	3.6W
Fuerza y golpe	Fuerza y golpe
Tamaño del cuerpo	30x16x15mm
Tamaño de barra de émbolo	6x58mm

Ilustración XIV Solenoide 12V Efecto Lineal 3.7×1.5mm JF0530B

Grabador de escritorio DIY CNC Láser [21]

Propiedades	**Descripción**
Nombre del producto	Grabador de escritorio DIY CNC Láser
Material	Acero inoxidable y acrílico
Tensión de trabajo	DC 12V
Potencia del láser	500mW
Área de grabado	65 × 50 cm

Ilustración XV Escritorio de 500 mW DIY Láser Grabadora Máquina de grabado de madera Marcado de imagen Impresora CNC 65x50 cm

Interruptor de final de carrera para Arduino [22]

Propiedades	**Descripción**
Nombre del producto	Final de carrera mecánico, Endstop
Características	Interruptor 1A 125V AC
Longitud de cable	70cm
Dimensiones del PCB	40mm x 15m

Ilustración XVI Final de Carrera Mecánico CNC

Cinta adhesiva de cobre

Propiedades	**Descripción**
Nombre del producto	Cinta de lámina de cobre
Material	cobre + pegamento
Tipo	dos
Tamaño	10mm x 25 m (a x L)
Peso	95g

Ilustración XVII cinta adhesiva de cobre[23]

Jumper wires

Propiedades	**Descripción**
Nombre del producto	Jumper wire
Longitud	20cm
Colores	variados
Material	plástico + cobre estañado + punta acero

Ilustración XVIII Jumper wires[24]

Cable de cobre

Propiedades	**Descripción**
Nombre producto	100 metros cablecillo <<Ariston>>
Material	Cobre estañado, Cubierta P.V.C. ø
Medidas	Exterior 1.4mm, sección 0.25 mm^2

Ilustración XIX Cable de cobre

Placa Arduino

Propiedades	**Descripción**
Nombre	Placa Arduino
Dimensiones	105 mm x 102 mm x 2mm
Número controladores	Cuatro controladores de motor paso a paso
Amperaje controladores	2.5 amperios
Tipo motores	Mayoría, hasta NEMA23 y algunos motores NEMA34

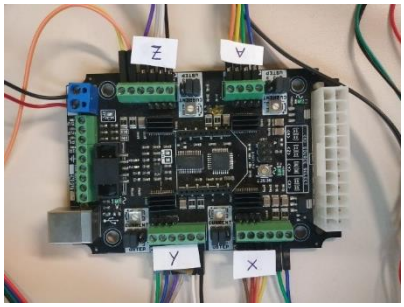

Ilustración XX Placa Arduino

Buda ball

Propiedades	**Descripción**
Nombre	Buda Ball, flyte
Dimensión	Esfera 20mm, Base: 110 mm x 110 mm x 23 mm
Color	nuez
Componentes	bobinas, imanes, sensores hall, resistencias…

Ilustración XXI Levitador buda ball

Webcam

Propiedades	**Descripción**
Nombre	HD web camera Loetad
Resolución	1080p
Conector	USB
Lente	gran angular

Ilustración XXII Webcam

USB tipo B

Propiedades	**Descripción**
Nombre	USB 2.0 tipo B
Descripción	Cable USB 2.0 con conector tipo A macho a conector tipo B macho, calibre 28 AWG, soporta hasta 480 Mbps
Material	Cobre
Protección	film aluminio, conectores bañados en níquel
Longitud	100cm

Ilustración XXIII USB tipo B

Conector Jack de alimentación hembra

Propiedades	**Descripción**
Nombre	Jack hembra eléctrico
Dimensiones	5 x 2.1mm
Corriente	5 A
Material del Contacto	Cobre berilio, latón, bronce fosforado
Tensión Nominal	12 V dc

Ilustración XXIV Conector jack de alimentación hembra

Fuente alimentación

Propiedades	**Descripción**
Nombre	Fuente de alimentación PeakTech 6255 A
Dimensiones	25.5 x 7.8 x 16 cm; 1.4 kilogramos
Output	0-31V 0-5 A
Resolución	10 mV y 1 mA

Ilustración XXV Fuente de alimentación

Fuente alimentación del Arduino

Propiedades	**Descripción**
Nombre	AC/DC Adapter
Input	100-240V~2.0A 50-60Hz
Output	: 12V 5.0A

Ilustración XXVI Fuente de alimentación Arduino

Extras

Aparte de los materiales mencionados, en el proyecto también se ha utilizado madera para la construcción de la caja, un trípode para sujetar la cámara, papel de aluminio para la colocación de la mascota y las herramientas para cortar, soldar o pegar. Además de esto, cabe mencionar los materiales utilizados pero descartados para versión final, como distintos levitadores o un motor para el eje-Z.

Levitadores

Probamos 3 levitadores magnéticos, como posibles mascotas. Dos de ellos fueron comprados ya montados y un último lo montamos desde cero.

Ilustración XXVIII Primer levitador probado

El primer levitador consiste en un disco que levitaba sobre una base de metal que esconde los circuitos que permiten que levite, tiene 4 luces led para facilitar la colocación del mismo.

Ilustración XXVII Segundo levitador probado

El segundo levitador, es una esfera que levita sobre una base de madera. Este lo hace ligeramente más alto que el primero, además las personas a las que se les preguntó, decían sentir más apego hacia la bola que hacia el disco.

Compramos el tercer levitador para intentar añadir algo más de altura a la mascota, pero, a pesar de que esta altura era ligeramente mayor (igualmente la diferencia era a penas apreciable), conseguir que se mantuviera en equilibrio fue prácticamente imposible.

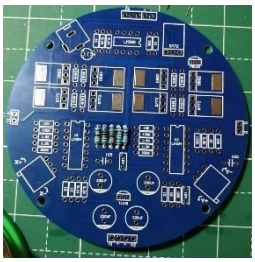

Ilustración XXXI Tercer levitador sin soladar

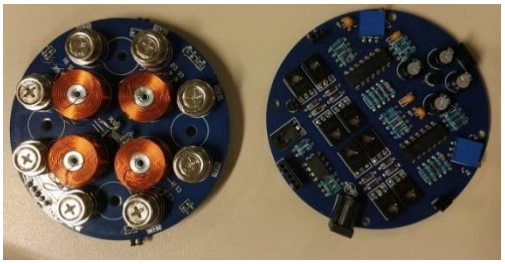

Ilustración XXX Tercer levitador soldado

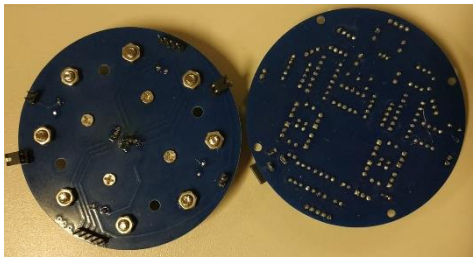

Ilustración XXIX Parte trasera del tercer levitador soldado

Para elegir el definitivo, tuvimos en cuenta en primer lugar la estabilidad. La que ofrecían los dos primeros era alta y similar, pero la del tercero era considerablemente inferior, por lo que este quedó descartado.

Para remediar la poca altura que conseguimos con los dos primeros levitadores, intentamos añadir imanes bajo el sistema, pero no fue efectivo en ninguno de los dos casos. Al no tener factores objetivos para quedarnos con un levitador u otro, decidimos quedarnos con el segundo por la mayor conexión que sentían las personas con él.

Ejes del proyecto

Una vez elegida la mascota, teníamos que decidir de qué manera íbamos a conseguir que se moviera. Hubo dos ideas principales:

- que el levitador estuviese sobre algo similar a una *Roomba* y que se pudiera mover libremente por una superficie plana.
- que el espacio estuviese delimitado, montarlo sobre unos ejes y tapar estos ejes para conseguir la ilusión de movimiento independiente por parte de la pelota.

La primera opción presentaba problemas del tipo de reconocer el espacio previamente al movimiento, que pudiese chocar con objetos inesperados y el principal que es que se vería claramente que está levitando por estar encima de una base. Esto rompía mucho la ilusión de que la mascota era una pelota capaz de volar y se convertía más en una *Roomba* con una pelota encima. En el segundo caso esta ilusión se mantenía.

Por ello, a pesar de que con la segunda opción la mascota tan solo se podría mover por un espacio reducido, elegimos este camino.

Una vez acordado el método a utilizar para el movimiento de la mascota, había que montar el sistema que permitiese esto. Para ello, montamos la máquina de grabado CNC mencionada anteriormente.

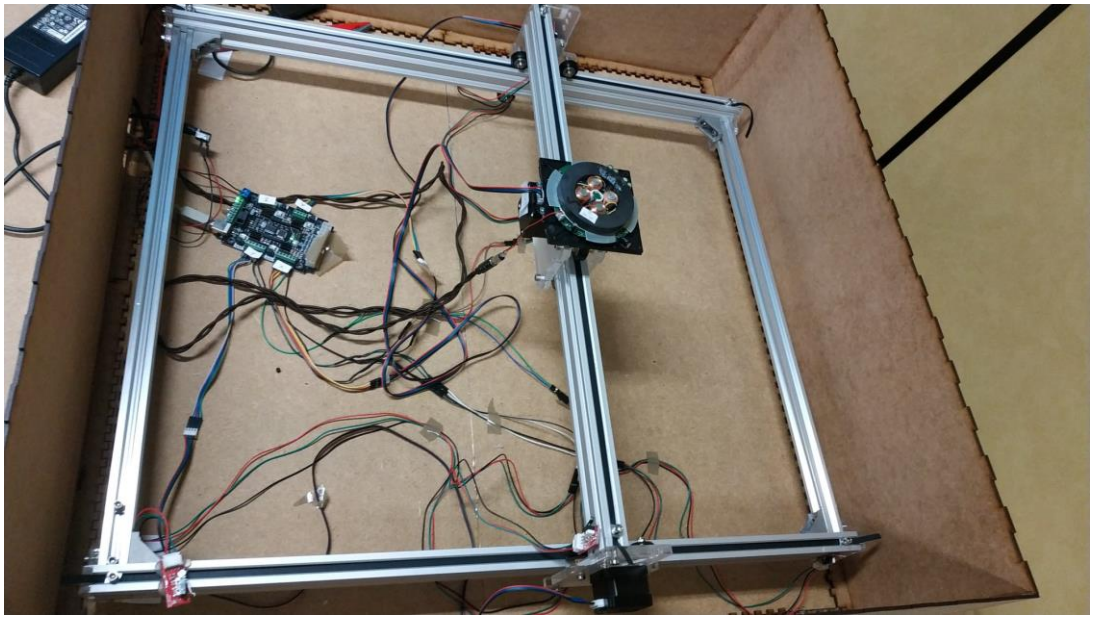

Ilustración XXXII máquina CNC con el levitador encima

La máquina consta de tres motores que se conectan al controlador. El controlador manda a cada motor el sentido en el que tienen que girar y a qué velocidad y durante

cuánto tiempo para llegar a la posición destino. Hay que tener en cuenta que el motor A es el espejo del Y, por lo que hay que conectarlo de manera inversa (están mirando en sentido opuesto).

El controlador se encarga de mandar a A y a Y la misma instrucción para que se puedan mover en paralelo.

El eje X era en el que se encontraba el láser de la máquina para el grabado. Nosotros no vamos a necesitar esto, por lo que no lo montamos y en su lugar, en la parte superior del motor colocamos la base del levitador.

Hay que mencionar que reemplazamos la placa que venía con el kit de montado de la máquina porque no disponía de pines para conectar finales de carrera. Estos sirven para hacer homing (instrucción de la máquina para, en nuestro caso, ir a la parte superior izquierda y saber así en qué posición se encuentra la mascota para poder empezar el ciclo) y como medida de seguridad para en caso de mandar una posición de destino que se salga del límite de la máquina, se activen[1].

Se necesitan cuatro finales de carrera porque se puede salir de los límites tanto en el eje X hacia el positivo y el negativo, como en el eje Y en los mismos sentidos, por lo que los colocamos para evitar el choque del motor X y del Y. No es necesario usar más para el motor A, ya que este se mueve a la vez que el Y, por lo que, si el motor A fuese a chocar, el Y también lo haría.

Por otro lado, los dos finales que se encargan de avisar si ha habido choque en el eje Y, están conectados en paralelo, ídem con el eje X. Esto es porque en nuestra placa tan solo teníamos opción para conectar un final de carrera por motor (a excepción del A por las razones ya explicadas) y realmente no nos importa saber si se ha chocado hacia un lado o hacia otro, simplemente queremos que parase la ejecución para poder volver a hacer homing y revisar el error que haya podido causar que se mandara una posición fuera del alcance de la máquina.

[1] El final de carrera es un interruptor. Si, por ejemplo, mandamos un destino que fuera demasiado alto hacia lo positivo en el eje x, el motor chocaría con el final de carrera moviendo la palanca de este. Esto haría contacto y mandaría la señal de ALARMA al controlador. Parando así todos los movimientos.

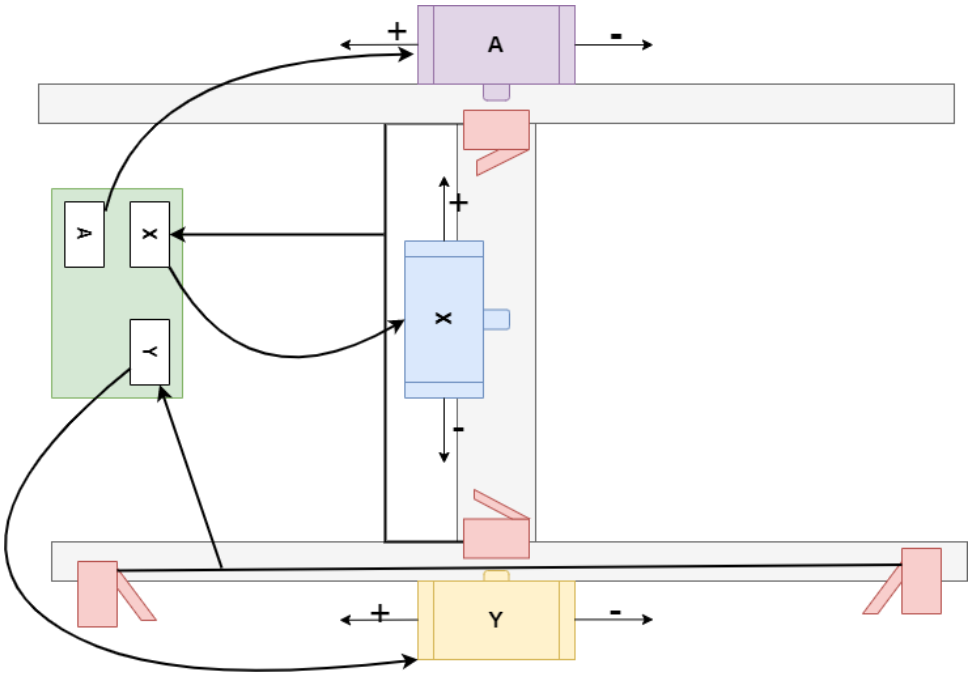

Ilustración XXXIII Esquema máquina CNC

Por otro lado, el controlador se conecta a una fuente de alimentación de 12V y 5A. Y al ordenador mediante un cable USB.

Xilófono

Una vez tenemos establecida la parte principal del proyecto, podemos pasar a los añadidos. Vamos a hablar sobre cómo funciona el xilófono del proyecto.

Ilustración XXXIV Xilófono con solenoides y cintas de cobre

El xilófono está compuesto por 5 teclas pero la mascota no las puede tocar directamente porque estas son magnéticas y quedaban atraídas magnéticamente a la esfera.

Por ello, pensamos en una solución que no involucrara el contacto entre el xilófono y la mascota. Ahora, delante de cada una de las teclas hay situado un solenoide y en frente de cada uno estos hay dos tiras de cobre.

Cuando la mascota toca las dos tiras correspondientes a un solenoide a la vez, este se activa y se dispara, chocando así con la tecla y emitiendo el sonido deseado.

Pero, ¿cómo funciona esto?

Pues bien, el mecanismo es sencillo, pero hubo que tener unos pequeños detalles en cuenta.

Lo primero es que los cables que habíamos utilizado para el resto del hardware eran magnéticos y eso era un problema siendo que nuestra mascota es un imán, porque como pudimos observar, se atraían, y cuando pretendía moverse de nuevo, la mascota se desequilibraba y chocaba contra el suelo. Por ello, decidimos utilizar cable de cobre.

Una vez solucionamos este problema, agujereamos la tabla en los lugares necesarios para poder tener los cables por debajo y que el exterior se viese más ordenado. El siguiente problema se originó cuando una vez estuvo todo en su sitio, probamos a encender el controlador de los ejes y la base del levitador arrancó los cables. Esto sucedió porque está tan cerca de la tapa (para poder sacar el máximo partido de la poca altura a la que puede levitar) que incluso unos cables de un milímetro de grosor era

demasiado. Por ello, cambiamos la parte de los cables que pasaban por la parte a la que el movimiento de los ejes puede llegar (la caja de madera es más grande que los ejes para que haya espacio suficiente para los motores y para poder añadir objetos en el playground a los que la mascota no pueda llegar, como las teclas del xilófono) por cinta adhesiva de cobre.

Cuando solucionamos esto, volvimos a probarlo y la base del levitador rasgó algunas de las cintas de cobre, así que decidimos ponerles cinta aislante por encima para evitar que si en alguna ocasión la base del levitador araña esa parte, la cinta no se rompa. Hay que mencionar que la cinta de cobre es frágil y que por ello este tipo de roces desgarró el material pero, con la cinta adhesiva encima no hay riesgo.

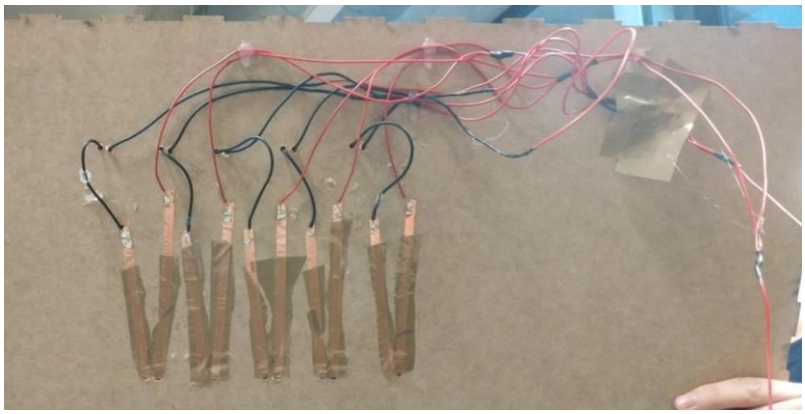

Ilustración XXXV interior de la caja con el mecanismo para el xilófono

Como hemos dicho, el circuito es algo sencillo. El solenoide se activa cuando por uno de sus cables recibe 12V y 300mA y por el otro está en tierra. Al probar con diferentes valores, decidimos que los óptimos eran 12V y 700 mA. Como no queremos tenerlo continuamente activado, si no que cuando algo suceda se active y choque con la tecla, dejamos el circuito abierto. Así que, conectamos uno de los cables a tierra y el otro lo dejamos libre. Hacemos esto para cada uno de los solenoides. Pero ahora necesitamos una forma de cerrar el circuito.

La solución evidente sería poner un interruptor que esté abierto por defecto y que, cuando esté pulsado, cierre el circuito y con ello se dispare el solenoide. Pero nuestra mascota, además de ser un imán, tiene otra peculiaridad: conduce electricidad. Por ello, la misma mascota puede cerrar el circuito tocando a la vez las dos cintas de cobre de una tecla.

Así que, unimos el cable del solenoide que teníamos suelto a una cinta de cobre y cogemos otro cable para conectarlo a la parte positiva de la fuente de alimentación al

que previamente también le hayamos añadido una cinta de cobre. De nuevo, tenemos que hacer esto para los cinco solenoides.

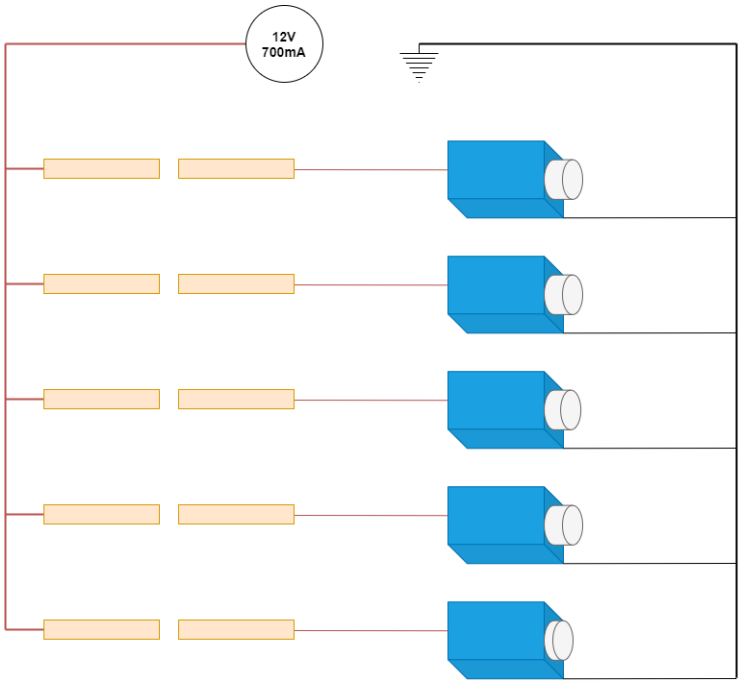

Ilustración XXXVI Diagrama xilófono

SOFTWARE

El desarrollo de software se ha realizado a dos niveles. Por un lado, como se ha comentado previamente, ha sido necesario programar en un entorno 3D un simulador que ayudara en la fase del diseño del sistema interactivo LevPet. Para ello se ha programado un simulador en Unity.

Por otro lado, se han programado los siguientes 3 módulos software que permiten dar al sistema LevPet la funcionalidad deseada.

- El detector de emociones y de posición de la cara de la persona usuaria
- El detector de posición de la pelota de ping-pong
- El módulo que permite gestionar la interacción entre la mascota y la persona
    - base de la comunicación con la máquina

Los dos detectores están escritos en Python y se comunican con el simulador mediante el protocolo OSC.

Detector de posición de pelota de ping-pong

Este es el script encargado de detectar la posición de la pelota de ping-pong en la tabla y mandárselo al simulador. Disponemos de una webcam que colocamos encima de un

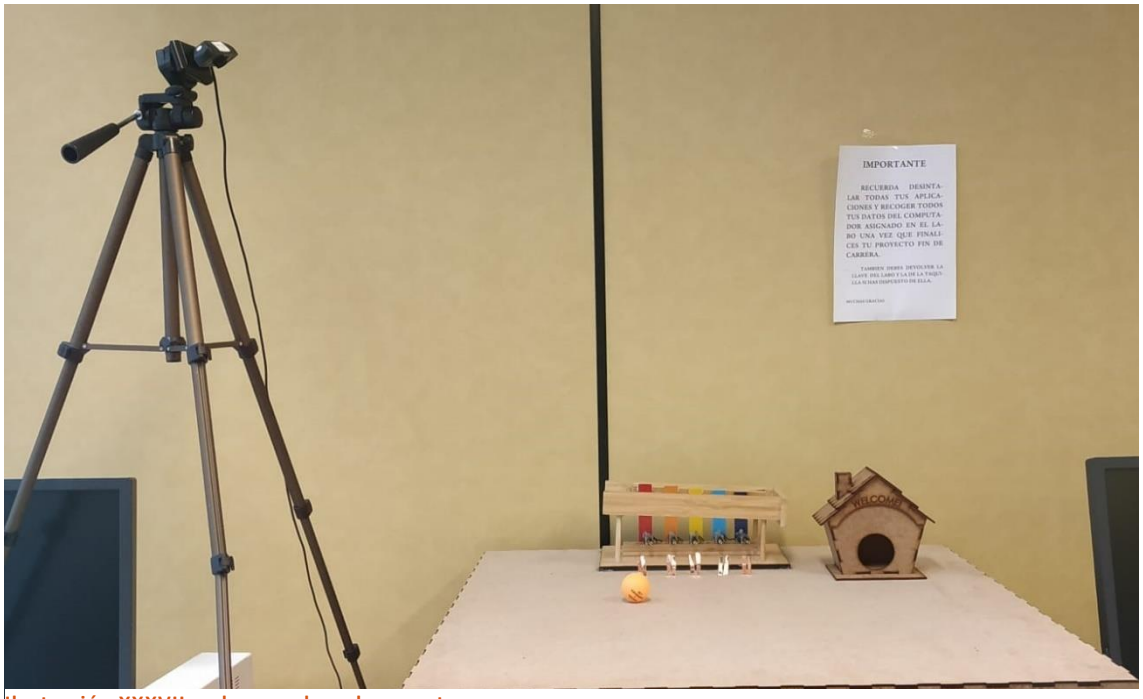

Ilustración XXXVII webcam sobre el proyecto

trípode para que la imagen pueda abarcar toda la superficie por la que nuestra mascota se pueda mover.

Para detectar la pelota, hacemos una máscara de color naranja para distinguirla del resto de elementos. Esto lo conseguimos definiendo un rango de valores HSV que corresponden a los colores de la pelota, con un pequeño margen para que los cambios de luces, si no son demasiado drásticos, no afectaran a la detección.

Para encontrar la pelota, aplicamos un filtro Gaussiano[2] con el objetivo de reducir el ruido que pueda tener la imagen, después, pasamos la imagen a colores HSV. Una vez tenemos la imagen dividida en colores, nos quedamos con los píxeles que pertenecen al rango de colores naranja que hemos definido para la pelota.

Tras ello, erosionamos y dilatamos la imagen para eliminar pequeñas manchas que puedan seguir en la imagen. A continuación, buscamos los contornos que quedan en la

Ilustración XXXVIII frame con la pelota detectada

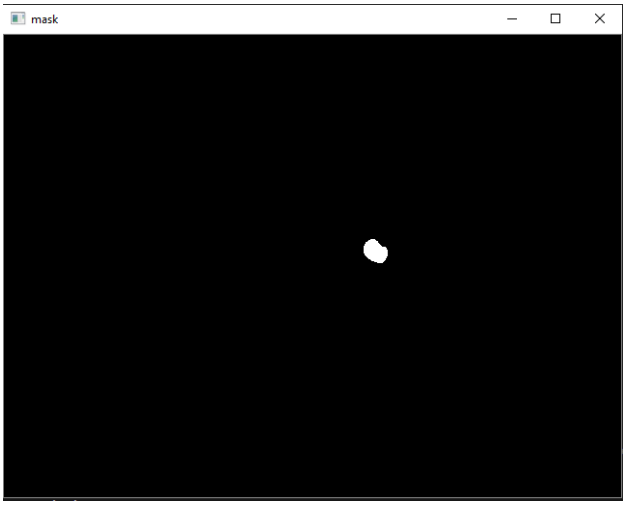

Ilustración XXXIX máscara de color para detectar la pelota naranja

imagen y, si hay al menos uno, nos quedamos con el más grande y creamos el círculo más pequeño posible que rodee el objeto (en amarillo en la imagen). De este círculo, guardamos la posición central, que es la que nos dice la posición en la que se encuentra la pelota en coordenadas de la cámara.[26]

Como podemos observar, nuestra webcam no es cenital, por lo que, aunque hiciésemos una transformación minmax de rango de píxeles de la imagen de la cámara a rango de posiciones del proyecto, las coordenadas obtenidas serían irreales. Además, aunque fuese cenital, seguiría existiendo la deformación de la imagen (sobre todo en los bordes) debido a la lente.

[2] Este tipo de filtros Gaussianos para reducir el ruido, actúan como si se tratase de un suavizado de la imagen.

Por ello, tuvimos que realizar un cambio de base para obtener la posición deseada para el proyecto. Para conseguir esto, tenemos que definir a qué rango queremos que nos pase las coordenadas, en nuestro caso la anchura la queremos de 0.2 a 8.5 y la altura de- 0.2 a -7.5 porque estas son las coordenadas por las que se mueve la mascota en el simulador, y definimos la parte de la imagen de la cámara que se corresponde a ese espacio. Para ello, marcamos en la imagen los cuatro puntos que delimitan el área del playground por el que se puede mover la mascota (en rojo en la imagen).

Tras ello con la función de getPerspectiveTransform de openCv, calculamos la matriz de cambio de base y, a continuación, le pasamos el punto que hemos obtenido desde la cámara y nos devolverá las coordenadas del proyecto.

A la hora de comunicarnos con el simulador, como hemos comentado antes, utilizamos OSC. Para establecer la conexión, primero tenemos que abrir un puerto para la comunicación UDP en local. A continuación, con la instrucción

```
client.send_message("/pingpong",mensaje)
```

mandamos mensaje “noBola”, si es que no se ha detectado ningún contorno, o bien las coordenadas de la pelota, si es que se ha detectado. Para no enviar mensajes que no aporten nueva información, tan solo mandamos la posición de la pelota si esta ha variado más de un rango definido.

Detector de emociones y de posición de la cara

Para este apartado, hemos utilizado una librería ya establecida[27]. Ella detecta la emoción expresada y dibuja un recuadro alrededor de la cara. A partir de ahí, sacamos el punto central del recuadro para tener una posición exacta de la cara en la imagen.

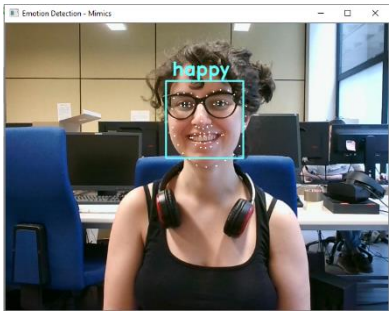

Ilustración XL Detección cara feliz

Una vez hemos obtenido este punto, tenemos el mismo problema que en el caso anterior, que el rango no es el mismo.

En este caso no nos interesa saber la altura de la cara, tan solo la posición a lo ancho de la imagen. Por eso mismo, podemos ignorar el problema de cambio de base.

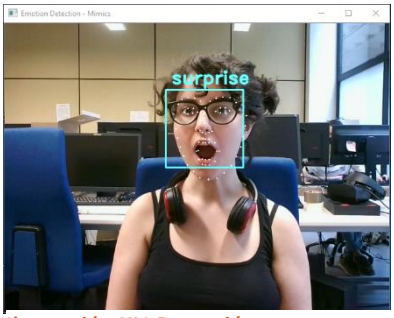

Ilustración XLI Detección cara sorprendida

Para mandar la información, tenemos que abrir un puerto para la comunicación UDP en local. Desde este script tenemos que mandar dos informaciones diferentes. La emoción y la posición, por lo que según lo que queramos mandar, utilizamos:

```
client.send_message("/cara",mensaje)

client.send_message("/emocionDetectada",mensaje)
```

De nuevo, para evitar enviar información repetida, solo mandamos el mensaje si la posición o la emoción ha variado.

Simulador en Unity

El proyecto comenzó con la realización del simulador para poder continuar sin tener que meternos en el hardware, pero, conforme fuimos avanzando, vimos que lo hecho en la simulación podía ser útil para la comunicación con el hardware.

Empezaremos explicando en primer lugar, la creación de los elementos básicos de la escena: el suelo, la mascota y su casa.

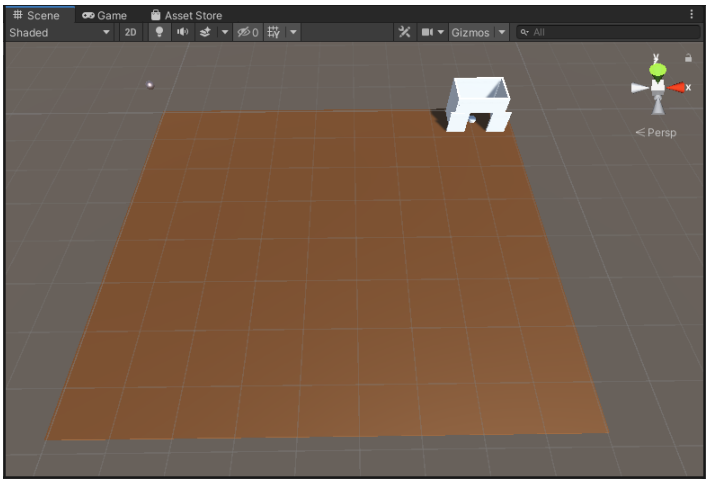

Ilustración XLII Escena en Unity con los elementos básicos

Elementos básicos de la escena

Queremos respetar las proporciones del mundo real en la escena, en este proyecto, 1 unidad de Unity equivale a 0.1 metros en la realidad.

Para la mascota, creamos un GameObject, en concreto, una esfera (3D object). Le asignamos un material para que se asimile a la mascota real y establecemos sus medidas como 0.2x0.2x0.2. Además, colocamos su posición de origen en la esquina superior izquierda, ya que ese es el lugar al que va la mascota real al hacer homing.

Para la casa, creamos un cubo para cada pared y los hacemos hijos de un GameObject vacío que representa la casa. De esta manera, si en un futuro queremos modificar su posición, podemos mover toda la casa a la vez. Aunque la casa real no es blanca, hemos querido representarla así para que destaque más sobre el suelo. Además, no nos importa que la casa no sea igual que la de madera, ya que lo único importante es el ancho del hueco de la entrada y el lugar de las paredes.

Para el suelo, simplemente hemos creado un plano con las medidas ligeramente mayores a las que tienen los ejes del proyecto por razones que comentaremos a continuación.

Definiendo el espacio de movimiento con Navmesh

Para el movimiento de la mascota, hemos usado el NavMesh de Unity. Esta herramienta nos permite definir el área por el que se podrá mover un agente, en nuestro caso, la mascota. Para ello, seleccionamos el suelo, y en navegación, marcamos la opción de estática, además, definimos el área como Walkable. Con esto indicamos que, si no hay ningún obstáculo, la esfera sería capaz de desplazarse por todo el plano. Una vez terminamos estos pasos, en la pestaña Bake, elegimos la opción Bake. Pero, podemos observar cómo los bordes del suelo no se consideran como parte navegable.

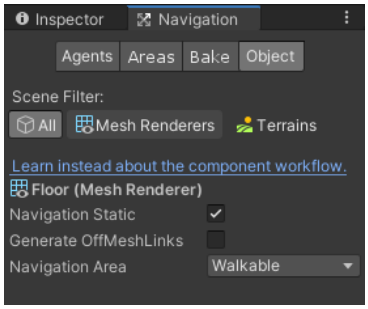

Ilustración XLIII Pestaña navegación del suelo

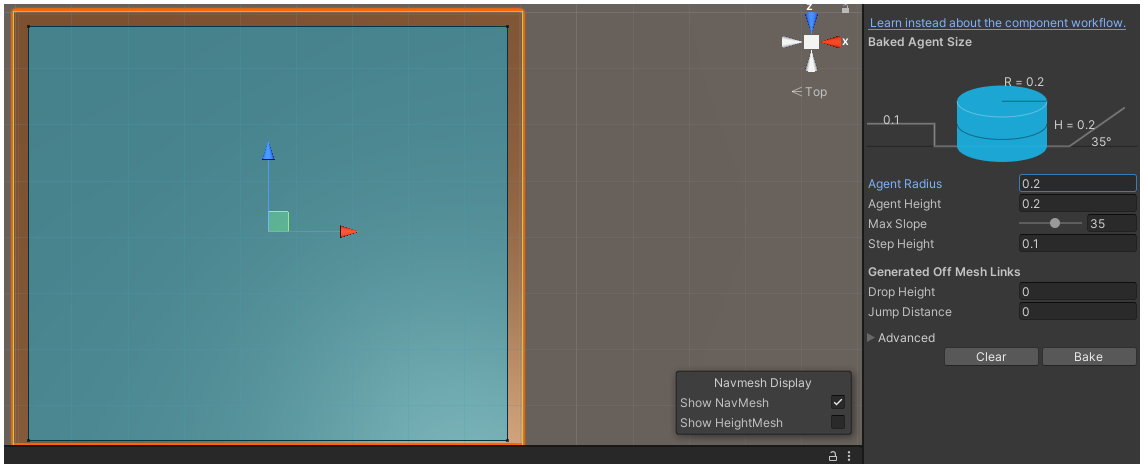

Ilustración XLIV Suelo con NavMesh

Esto es porque el agente tiene radio y altura, y el NavMesh trabaja con el punto central del GameObject. Así que aunque podríamos cambiar estos valores para que pudiese navegar por todo el espacio, decidimos no solo mantenerlos, si no aumentarlos para que pueda esquivar obstáculos más holgadamente.

A la hora de hablar de obstáculos en nuestro proyecto, nos referimos a las paredes de la casa. Si no definimos que las paredes son un obstáculo, el sistema entenderá que se pueden atravesar. Para evitar que esto ocurra, añadimos a los componentes de la casa la opción de Nav Mesh Obstacle. Para que el NavMesh tenga en cuenta estos elementos, tenemos que seleccionar la opción de Carve.

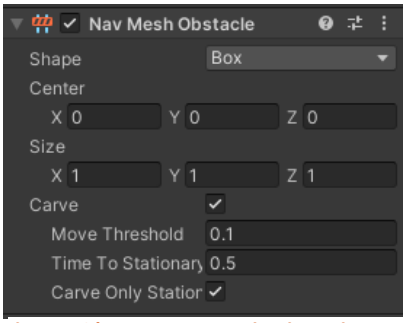

Ilustración XLV Nav Mesh Obstacle para la casa

De esta manera, quedaría definido el espacio por el que se puede desplazar la mascota. Si ahora le mandásemos como destino un punto que no pertenece a esta área, la mascota se desplazaría hasta el lugar más cercano dentro del espacio permitido y se

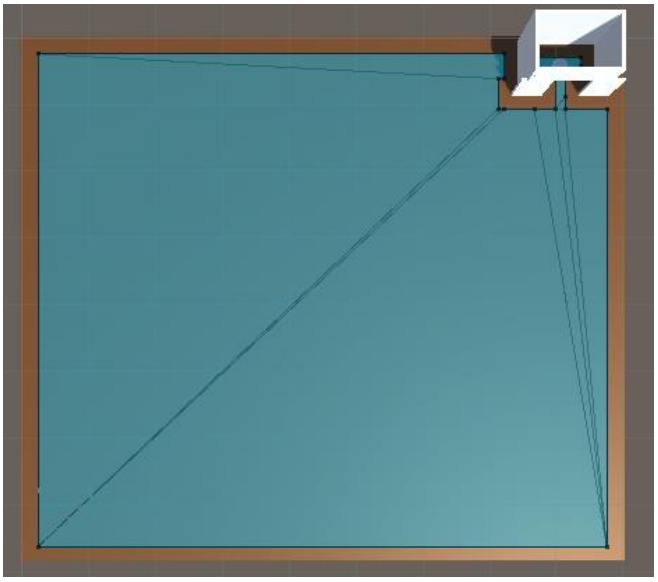

Ilustración XLVI Estado final del NavMesh

quedaría parada. Nos interesa que esto suceda ya que con el hardware, no podemos sobrepasar los límites que están definidos por los ejes.

Consiguiendo el movimiento con NavMeshAgent

Una vez tenemos definido el espacio por el que puede moverse la mascota, necesitamos establecer de qué forma vamos a conseguir que se mueva. Esto es sencillo mediante el uso del NavMeshAgent. Así como el NavMesh nos facilita delimitar el espacio, el NavMeshAgent nos ayuda con el movimiento de un agente que quiera desplazarse dentro de esta área.

Ilustración XLVII componente NavMeshAgent de la mascota

Para ello, en primer lugar, tenemos que asignarle a nuestra mascota el componente NavMeshAgent. Esto nos permitirá utilizar la base del movimiento de la mascota, la función SetDestination(). Esta función nos permite definir un punto del espacio y el agente se moverá hasta dicho punto (o hasta el lugar más cercano que pertenezca al espacio del NavMesh) siguiendo el camino óptimo y esquivando los objetos.

Podemos observar en la imagen los parámetros que definen este movimiento, así como el camino a seguir. A nosotros nos interesa el atributo Speed ,ya que cuando pensamos en lo que más identifica un movimiento, a parte de la fluidez, lo primero que nos viene a la mente es la velocidad del mismo.

Por ello, para los distintos movimientos que realizará la mascota, la velocidad será uno de los valores más importantes a tener en cuenta.

Ahora que sabemos cómo ir a un destino, tenemos que pensar cuáles son los adecuados. Por lo que, creamos el Script MoveRandomly y se lo asignamos a la esfera. Este Script será el que se encargue del movimento de la mascota.

Como no queríamos que el movimiento estuviera predefinido, tratamos de establecer unos destinos aleatorios, así que, cuando la mascota llegaba a dicha posición, se calculaba un nuevo destino y se dirigía a él. Nos dimos cuenta pronto de que con esto no bastaba, ya que si el punto origen se encontraba en un lado del área y el punto objetivo en el otro, el movimiento era demasiado lineal y artificial.

La solución fue establecer un tiempo para actualizar la posición objetivo aunque no hubiese llegado a ella. Así que, ahora, el movimiento está definido por la velocidad y el tiempo que pasa entre ir en una dirección hasta cambiar a otra.

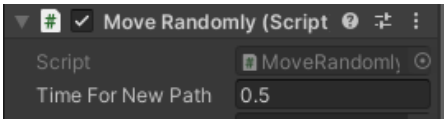

Ilustración XLVIII Variable Time For New Path de Move Randomly

Para comprender la manera de programar esto, hay que saber primero que en Unity, se dispone de una función llamada Update() que se ejecuta cada frame. También existen corrutinas, que permiten ejecutar una función, en vez de de principio a fin en un solo frame, ejecutar una parte en un frame, devolverle el control a Unity, y en el siguiente frame continuar, esto es especialmente interesante en estructuras de bucle. Además, se puede añadir un retardo desde el momento en el que ejecuta la parte de la función correspondiente hasta que devuelve el control. Aunque nosotros no fuésemos a utilizar bucles, hemos querido implementar esta funcionalidad ya que puede permitir reducir la sobrecarga.

En nuestro caso, hemos sacado provecho de la parte del retardo, ya que si pusiéramos la instrucción de esperar en la función Update, no se podría seguir adelante con el resto del programa mientras estuviésemos esperando. Así que, nuestra corrutina consta una instrucción de espera de TimeForNewPath segundos; a continuación, la obtención de las coordenadas (x, 0, z) aleatorias que corresponden al nuevo punto objetivo (utilizando la función Random.Range, que devuelve un float entre un mínimo y un máximo específicados, para conseguir x e y) y, por último, una llamada a la función SetDestination() pasándole el destino de la mascota.

Comportamiento de la mascota

Mostraremos a continuación el diagrama que define el comportamiento que tendrá la mascota.

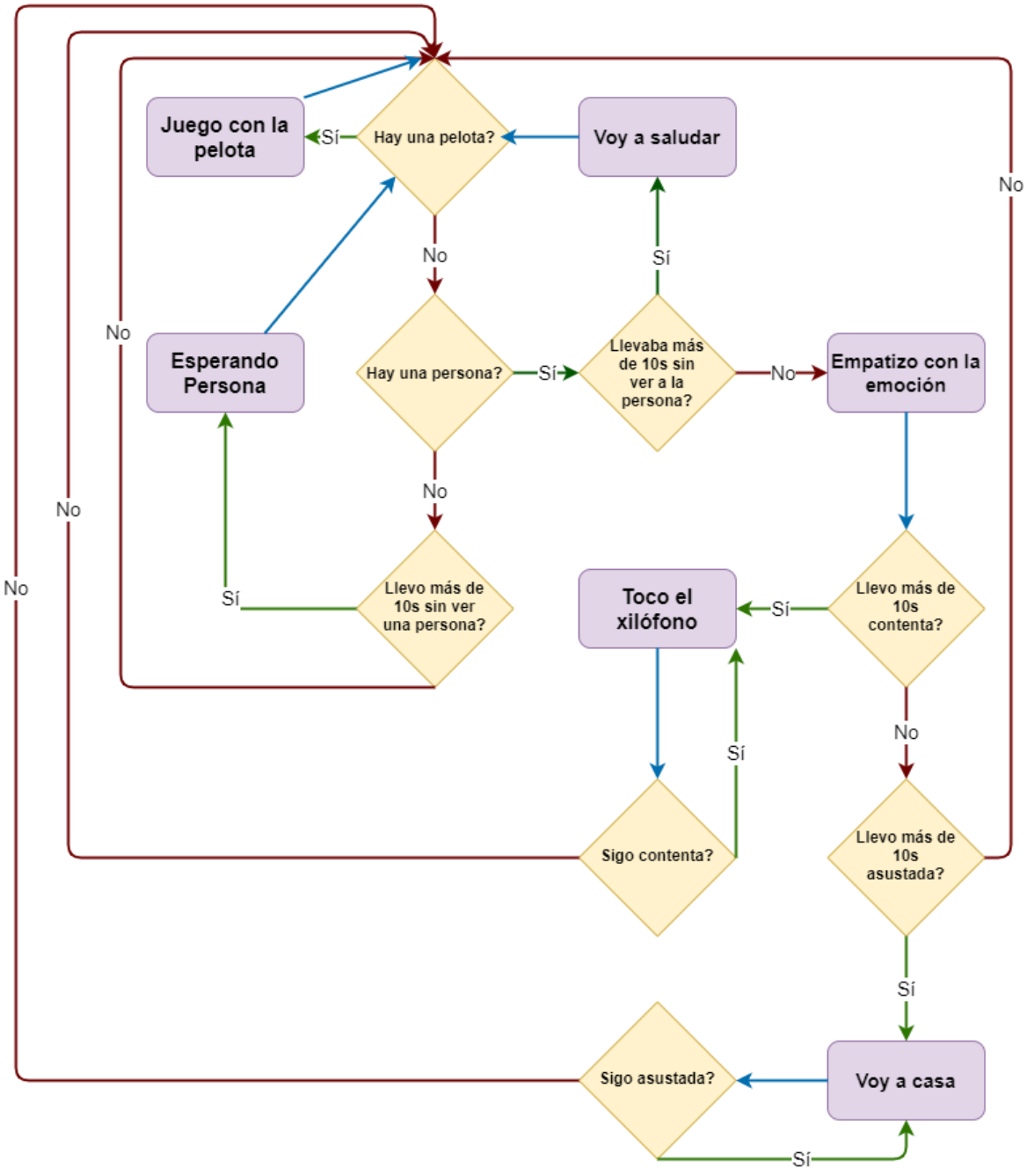

Ilustración XLIX Diagrama comportamiento mascota

Movimiento en relación con las emociones

En primer lugar, hemos creado tags con las emociones que pueden ser detectadas: Neutral (Neutro), Happy (Feliz), Surprised (Sorprendida), Sad (Triste), Disgusted (Asqueada), Anger (Enfadada) y Fear (Miedo), además, tenemos el tag de No face para representar que no se ha detectado ninguna persona.

En base a estas emociones, jugaremos con los parámetros que definen el movimiento que va a realizar la mascota.

En primer lugar, tenemos que distinguir las emociones que siente el usuario de las que siente la mascota. En la mayoría de las relaciones, la mascota imita lo que percibe, es decir, si el usuario está feliz, la mascota también lo estará. Al igual que si interactuásemos con un perro: si nosotros nos acercamos contentos, el perro actuará contento. Pero hay emociones que una mascota no imitaría si las ve en el usuario, por ejemplo, si la persona está enfadada, la mascota no se enfadaría, sino que se asustaría.

Tabla 1 Empatizar con las emociones

Emoción detectada en el usuario	Emoción de la mascota
Neutral	Neutral
Happy	Happy
Surprised	Happy
Sad	Sad
Disgusted	Sad
Anger	Fear
Fear	Fear

Una vez tenemos definido de qué manera va a empatizar la mascota con respecto a lo que siente la persona, vamos a definir los parámetros que identificarán el movimiento de cada emoción.

Para establecer estos valores, nos hemos basado en los movimientos de una persona cuando muestra estas emociones.

- El movimiento neutro es el movimiento por defecto o normal.
- Un movimiento alegre es más energético y rápido que uno normal, por eso, la velocidad establecida será mayor y el cambio de dirección será más frecuente sin llegar a ser excesivo.

- La tristeza es mucho más pausada y mucho menos energética, por lo que los movimientos serán considerablemente más lentos y el cambio de dirección será parecido al de un movimiento normal.
- Lo más característico y visual del miedo son los temblores. Cuando una persona experimenta terror o grandes niveles de ansiedad, su cuerpo activa todos los mecanismos que tiene para poder reaccionar mejor a la situación de peligro: el corazón se dispara para preparar al cuerpo para luchar, la respiración se acelera para oxigenar, aumentan los reflejos, … Esto hace que los movimientos sean rápidos y excesivos, por lo que la velocidad será alta y el cambio de movimiento será muy frecuente.

Tabla 2 Velocidad y cambio de dirección en función de la emoción

Emoción Mascota	**Speed**	**Time For New Path**
Neutral	1f	1s
Happy	1.5f	0.5s
Sad	0.25f	1s
Fear	1.5f	0.005s

Para programar esto, cuando vamos a establecer un nuevo destino, comprobamos cuál es el tag de la esfera (recordamos que esto representa la emoción del usuario y no de la mascota) y lo guardamos en una variable llamada emocionDetectada. Creamos una variable emoción y le asignamos el valor de la detectada; si esta es Sorpresa, Asco o Enfado, la cambiamos por la correspondiente. Una vez tenemos la emoción de la mascota, accedemos al parámetro navMeshAgent.Speed y a la variable timeForNewPath y le asignamos el valor que necesita. Si se detecta no face, la mascota seguirá con la emoción detectada previamente hasta el siguiente evento.

Actividades de la mascota

Esconderse/descansar

Cuando la mascota lleva más de un tiempo predefinido sintiendo miedo, esta va a su casa a esconderse, de la misma manera, si ha pasado mucho rato sin que tenga una interacción con una persona, se va a su casa a descansar.

Para poder conseguir esto, hemos puesto una cama. Para ello, creamos un GameObject cilindro y lo colocamos dentro de la casa. Ahora, cuando la mascota tenga que ir a la casa, se le asignará como posición de destino, la de la cama. Por otro lado, tenemos que

asegurarnos de que el NavMesh permita entrar a la casa y llegar a la posición del cilindro, así que, a la cama, no le asignamos la opción de obstáculo.

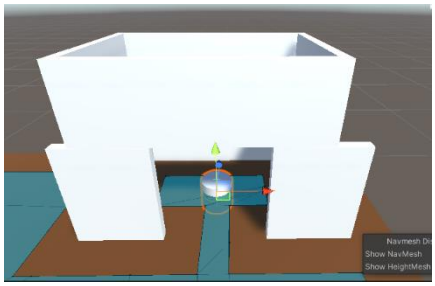

Ilustración L Casa con cama en el simulador

Una vez definida la posición a la que tiene que ir la mascota cuando quiere ir a casa, tenemos que pasársela al script que controla el movimiento del agente. Por lo que, creamos una variable pública de tipo GameObject y arrastramos el objeto que representa la cama al Script.

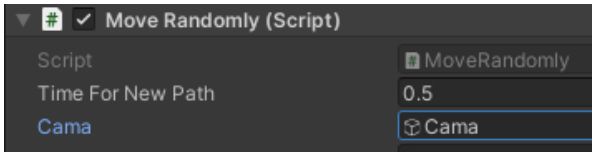

Ilustración LI Objeto cama en el script MoveRandomly

De esta manera, podemos acceder a las características de la cama, como la posición, desde el script MoveRandomly. La información sobre la posición se encuentra en el apartado Transform, así que para consultarla, tenemos que escribir cama.transform.position.

Ahora que tenemos la información del destino en el script, tenemos que calcular si el tiempo que ha transcurrido desde el inicio de la emoción de miedo es mayor que el estipulado para esconderse. Para ello, creamos una variable tiempoFear encargada de saber cuánto tiempo lleva asustada la mascota. Ahora, cuando asignamos el valor a la emoción, miramos si esta es miedo: si lo es, sumamos el tiempo transcurrido desde la anterior comprobación hasta ahora a la variable tiempoFear, si no, reseteamos el tiempo a 0. Comprobaremos en cada llamada a Update si hay que resetear o sumar. Como Unity funciona en frames en vez de en segundos, para añadir tiempo, hay que usar la opción Time.deltaTime, que devuelve los segundos que han transcurrido desde el anterior frame. Así, la actualización de este temporizador se haría mediante la instrucción tiempoFear += Time.deltaTime.

Ahora, a la hora de buscar un nuevo destino, si la emoción de la mascota es miedo, comprobamos si el temporizador es mayor que el límite establecido, si lo es, asignamos como destino del agente la posición de la cama.

Además, hemos comentado que también iría a casa para descansar, por lo que, si lleva más de un tiempo definido sin ver a una persona, se dirige a casa. Para ello, al igual que teníamos una variable temporizador para el miedo, creamos otra con el mismo propósito para no face que actualizaremos también en el Update cuando la emoción sea esta.

De nuevo, si a la hora de buscar un nuevo objetivo, la emoción detectada es no face, comprobamos si el temporizador de esta variable es mayor que el tiempo límite establecido. Si lo es, ponemos como destino la cama y, además, asignamos a true el valor de una variable booleana llamada esperandoPersona, que indica si la mascota está en casa esperando a que se aproxime un usuario para interactuar con él. Si se ha detectado cualquier emoción que no sea no face, esperandoPersona será igual a false. Esta variable será relevante cuando expliquemos la acción de saludar al usuario.

Jugar con la pelota

Una de las características principales de una mascota, son sus ganas de jugar. Un gato lo puede hacer con un plumero, un pájaro con un papel y un perro con un frisbee. Nosotros queremos que nuestra mascota juegue con una pelota de ping-pong.

Cada vez que haya una pelota en la escena, la mascota irá a jugar con ella moviéndola de un lado a otro. Para la simulación, hemos creado un GameObject esfera que representará la pelota.

Para añadir la pelota a la escena, basta con clicar sobre el punto donde queremos que caiga, entonces, la esfera aparecerá ligeramente por encima del punto seleccionado para caer con unas físicas que recuerdan a las que puede tener una pelota de ping-pong normal (grandes botes al principio de la caída y pequeños y rápidos al final).

Para conseguir este efecto, creamos un nuevo material de físicas. Esto nos permite cambiar los valores que definen las características físicas del material de un objeto. En nuestro caso, asignamos un valor de 0.85 al rebote, este número combinado con el resto de parámetros del material y, con el valor de Bounce Threshold de los ajustes del proyecto que hemos cambiado a 0.6, hace que la esfera se comporte como una pelota de ping-pong.

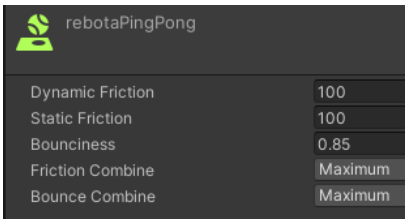

Ilustración LII Valores del material para la pelota de ping-pong

Tras definir el material, se lo asignamos al Sphere collider (encargado de manejar las colisiones de un objeto) de la pelota.

Ilustración LIII Material en el Sphere Collider

A parte, le añadimos el componente RigidBody[28] que, como el propio nombre indica, permite dar a los objetos las características de un cuerpo rígido. Así, podremos tratar al GameObject como si: tuviera masa, le afectara la gravedad, pudiera colisionar con otros objetos, ejercer fuerza sobre otros cuerpos, tener un coeficiente de rozamiento dependiendo de la superficie en la que se encuentren, tener velocidad lineal y angular, inercia y momento angular.

Además, añadimos este componente también a la mascota para que sea capaz de aplicar fuerza sobre la pelota y así, moverla.

Destacamos el parámetro Kinematic del RigidBody: si está activado, el objeto no será manejado por el motor de física, y puede solo ser manipulado por su Transform. Esto será de utilidad más adelante, cuando expliquemos la comunicación con los detectores.

Una vez hemos conseguido que la pelota se comporte como deseamos, pasamos a realizar el código para que, al clicar en una parte del suelo, esta caiga en ese lugar.

Para ello, creamos un nuevo Script al que llamaremos ActualizarPingPong[29] y se lo asignamos a la pelota de ping-pong.

En él, en la función Update, creamos un plano invisible horizontal. A continuación, creamos un rayo de dirección desde la cámara hacia el punto elegido por el ratón. Comprobamos si el plano y el rayo se han cruzado; en caso positivo, nos quedamos con la distancia desde origen del rayo hasta la posición de intersección, en caso negativo, no hacemos nada.

Por último, obtenemos la posición de la escena usando el método GetPoint que, dado un rayo y una distancia, devuelve el punto que, siguiendo el sentido del rayo, se encuentra a dicha distancia.

Este punto nos indica qué posición del suelo hemos elegido, por lo que, ahora, actualizamos la posición de la pelota a algo más arriba del punto para que, de esta manera, la pelota pueda caer. Lo conseguimos con la siguiente instrucción:

```
this.transform.position = new Vector3(worldPosition.x,origen.y,worldPosition.z);
```

worldPosition es la posición del punto elegido y origen es el punto en el que se encuentra la pelota antes de ejecutar el código.

Ahora que ya tenemos la pelota en la escena, necesitamos que la mascota juegue con ella. Para ello, asignamos como destino del NavMesh la posición de la pelota. Obtenemos esta posición de la misma forma por la que hemos conseguido antes la posición de la cama, creando una variable GameObject en el Script MoveRandomly y arrastrando el objeto pelota al campo. Como son dos GameObjects que actúan como objetos reales, es imposible que estén en la misma posición, así que, cuando la mascota va a la posición de la pelota, empuja a la misma, por lo que la nueva posición de la pelota no es la que era y así sucesivamente. De esta forma, conseguimos que la mascota mueva la pelota de un lado a otro.

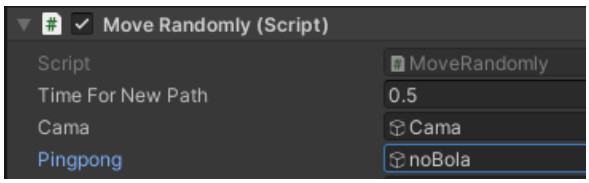

Ilustración LIV Script MoveRandomly con el parámetro de la pelota

Saludar al usuario

Para saludar al usuario, seguimos el mismo método que hemos utilizado para llegar a destinos concretos. Creamos un GameObject que represente la posición en la que se encuentra la persona. En este caso, un cilindro, pero como no nos interesa que sea visible, desactivamos la opción de Mesh Renderer (encargada de que el objeto se pueda ver en la escena).

Para poder acceder a esta posición desde el script MoveRandomly, volvemos a crear una variable pública GameObject a la que arrastraremos el objeto que representa al usuario.

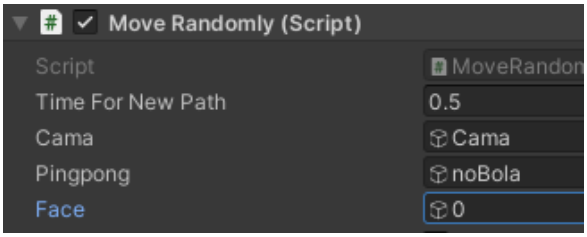

Ilustración LV Script MoveRandomly con el parámetro Face

Como en el proyecto real utilizaremos una única cámara para detectar la posición del usuario, solo nos interesa que este objeto se mueva por la parte baja del plano de la escena (que representaría la parte en la que podemos observar a la persona).

Para no estar continuamente yendo a saludar al usuario cuando este está presente en la cámara, utilizamos la variable esperandoPersona que hemos mencionado en el apartado de Esconderse/Descansar.

Cuando la variable esperandoPersona está a true y aparece un usuario, es decir, se detecta una emoción que no sea no face, la mascota se dirigirá al lugar de la escena donde se encuentre el usuario. Una vez esté ahí, esperará unos instantes y seguirá con el movimiento correspondiente a la emoción.

Tocar el xilófono

La última actividad que puede realizar la mascota, trata sobre tocar un xilófono.

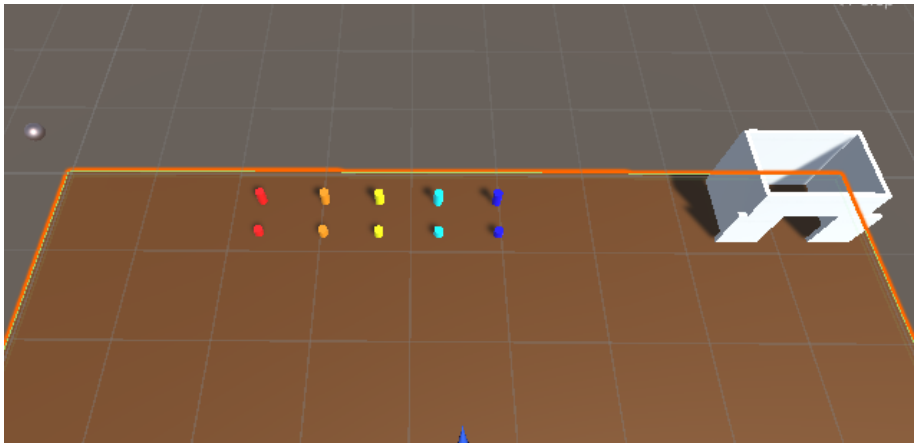

Ilustración LVI Xilófono en el simulador

Como podemos observar en la imagen, tenemos las teclas del xilófono repetidas. Esto es, porque la fila más alejada del borde es un paso previo al de las teclas. Si yo quiero ir a la tecla roja, tengo que pasar primero por la pequeña tecla de delante. Si no hubiésemos establecido estos puntos, a los que llamaremos preteclas, la mascota iría en línea recta entre las teclas, pasando continuamente por teclas que no corresponden.

Para poder acceder a la posición de estos elementos, hemos creados dos públicos de Transform (componenete del GameObject que contiene la información sobre la posición) en el Script de MoveRandomly. Aquí arrastramos las teclas y preteclas.

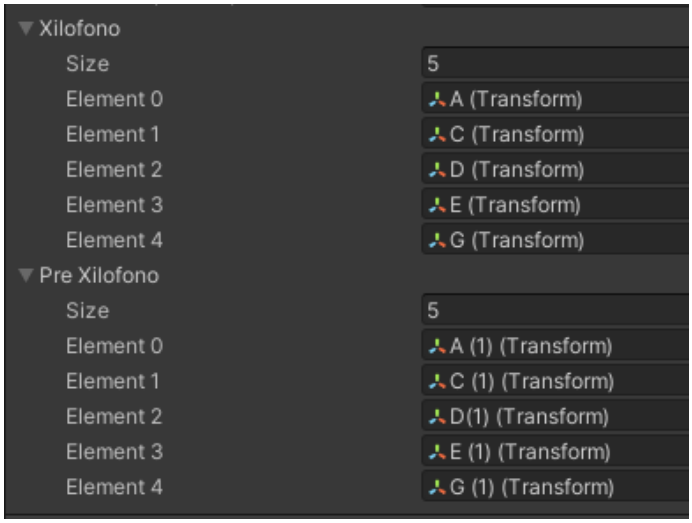

Ilustración LVII arrays de elementos del xilófono

La mascota pasa a la actividad de tocar el xilófono cuando lleva más de un tiempo sintiéndose feliz. El método es el mismo utilizado para obtener el tiempo que lleva asustada o sin detectar una cara.

Cuando se busca un nuevo objetivo, si se ha pasado del límite de tiempo a partir del cual va a tocar el xilófono, se elige al azar, mediante Random.Range(númeroDeTeclas), la pretecla que tiene que ir a tocar. Además, asignamos true a la variable destinoPreTecla, que nos informa de si se está yendo hacia una pretecla.

Utilizamos esta variable para que cuando acabe el tiempo de espera entre una dirección del movimiento y la siguiente, no cambie de dirección, si no que siga yendo hacia el destino establecido.

Si destinoPreTecla es true, comprobamos si la posición de la mascota es la misma que la de la pretecla (exceptuando la altura). Si lo es, cambiamos el destino a la tecla correspondiente y asignamos false a destinoPreTecla y true a destinoTecla.

DestinoTecla actúa de igual manera que destinoPreTecla con la diferencia de que, cuando se llega a la posición destino, tanto destinoTecla como destinoPreTecla, pasan a ser false. De este modo, se vuelve a buscar un destino. Si la emoción sigue siendo feliz, el nuevo destino volvería a ser una pretecla, de esta manera, seguirá tocando teclas hasta que se detecte otra emoción, se vaya el usuario o aparezca una pelota para jugar.

Conectando Unity y los detectores

Osc

Para la comunicación entre Unity y los detectores, usamos OSC. La parte del cliente, es decir, la información que mandan los detectores al simulador, ya la hemos explicado anteriormente. A continuación, entraremos en detalle en la parte del servidor.

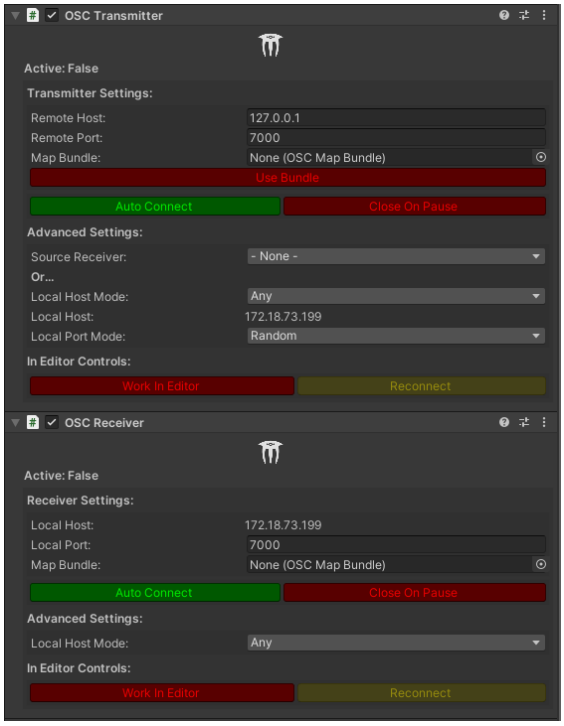

Ilustración LVIII OSCManager

Añadimos a Unity la librería de OSC [24] que nos permite acceder a las funcionalidades del protocolo. Una vez importada, tenemos que crear un objeto que sea el encargado de que sea posible la llegada y salida de mensajes, el OSC Manager. En él, abrimos un puerto, en nuestro caso el 7000 e indicamos que la interacción será en local poniendo la ip 127.0.0.1.

El Script OSC Transmiter, es el encargado de mandar los mensajes, y el OSC Receiver de recibirlos. Esto es suficiente para establecer la conexión, pero no es suficiente para enviar y recibir los mensajes donde queremos.

Para ello, tenemos que añadir al objeto deseado el Script que indica el tipo de dato que vamos a recibir. En el caso de la mascota, por ejemplo, recibiremos un String con la información de la emoción. Además, tenemos que indicar: a qué dirección llegará, recordemos que desde los detectores añadíamos un destino con /address; el puerto, que ya hemos estipulado que será siempre el 7000; el objeto que recibirá la información, que es el mismo que contiene el Script; y dónde recibirá la información, en nuestro caso, será en el tag de la mascota, así, cuando llegue el mensaje, se actualizará y podremos acceder a ella desde el Script MoveRandomly con

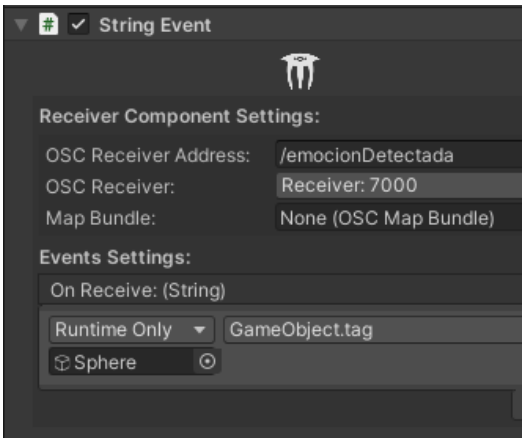

Ilustración LIX Script recibir emoción

emocionDetectada = gameObject.tag;

De manera similar, actualizamos la posición del usuario y de la pelota de ping-pong.

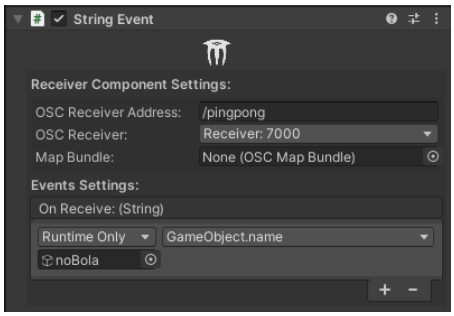

Ilustración LXI Actualizar ping-pong

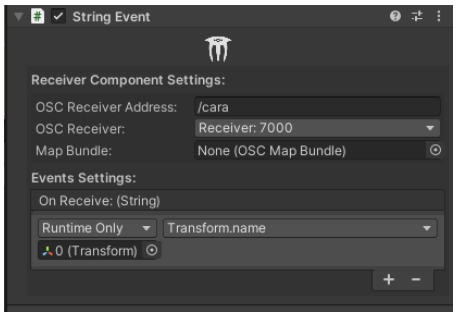

Ilustración LX Actualizar cara

Adaptación de las funcionalidades

En los apartados anteriores, hemos explicado cómo hemos usado el simulador para poder representar lo que en un futuro estará en la realidad. Ahora, comentaremos de qué forma hemos adaptado el simulador para que pueda recibir información de los detectores e integrarla en la escena.

PELOTA

En el caso de la pelota de ping pong, hemos creado una variable Físicas que indica si la pelota se va a comportar por físicas o si, por el contrario, se actualizará la posición en función de la detectada por la cámara. Para ello, si está deseleccionado, asignamos el valor true al parámetro Kinematic. Como ya explicamos, esto hace que el objeto deje de ser manejado por el motor de físicas y tan sólo puede moverse mediante el campo Transform.

Así, cuando nos llega información sobre dónde se encuentra la pelota real, asignamos el valor a transform.position y la simulada se transporta instantáneamente a dicha posición. Como la información sobre la nueva posición se manda cada vez que la pelota se ha movido más de unos pocos centímetros, da la sensación de movimiento continuo. Esto se realiza en un Script llamado ActualizaPing-Pong. Además de esto, necesitamos una forma de que aparezca o desaparezca de la escena. Cuando la cámara no ve la pelota, envía en vez de la posición, la palabra “noBola”. Si recibimos esto, desactivamos el MeshRenderer de la misma para que no sea visible.

Si desde el Script MoveRandomly vemos que esta cualidad está activada, consideramos que hay pelota en la escena y vamos a su posición. Por el contrario, si está desactivada, indica que no la hay y sigue con la actividad que estuviera realizando.

SALUDAR/DESCANSAR

En este caso, al no ser un objeto que contenga físicas, simplemente nos sirve con actualizar la posición cuando recibimos la nueva por cámara. Nuevamente, hacemos esto utilizando el transform.position. En el caso de este objeto, lo mantenemos siempre como no visible.

IR A CASA O TOCAR EL XILÓFONO

En este caso, no necesitamos hacer nada especial, tan sólo asegurarnos de alinear bien los objetos de la simulación con los reales.

Comunicación entre Unity y el controlador de los ejes

Por último, nos queda comentar de qué manera unimos el movimiento en la simulación con el movimiento de los ejes. Para ello, en primer lugar, importamos la librería [30] que nos ayudará con la comunicación por SerialPort (el controlador está conectado al ordenador mediante puerto USB). Esta librería dispone de un Prefab que contiene los Scripts para configurar el puerto y para mandar y recibir mensajes por este. Arrastramos el Prefab a la escena para poder acceder a estas funcionalidades, tras ello, configuramos con los parámetros que nos interesan, como el baudio a 112500.

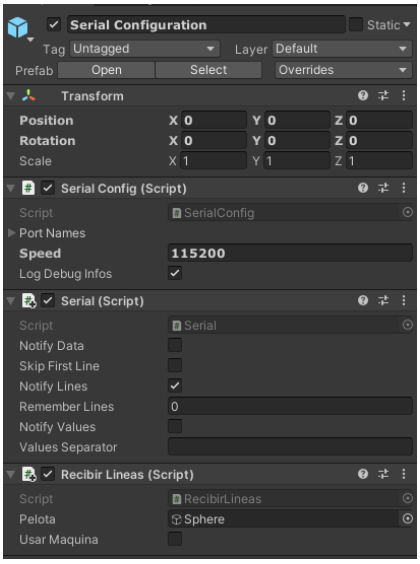

Ilustración LXII Serial Configuration

Además de los Scripts que vienen en el paquete, creamos uno llamado RecibirLineas para gestionar los mensajes que recibimos por el puerto. Dichos mensajes, informan sobre el estado de la máquina y pueden ser los siguientes:

- Ok: cuando la máquina ha terminado de ejecutar satisfactoriamente una acción
- ALARM: cuando ha habido algún problema
- [MSG:'$H'|'$X' to unlock]: cuando se necesita hacer homing

```
Script de Unity (1 referencia de recurso) | 0 referencias
public class RecibirLineas : MonoBehaviour
{
    // Start is called before the first frame update
    public GameObject pelota;
    private MoveRandomly elScript;
    private bool esperandoOkHoming;

    public bool usarMaquina;

    Mensaje de Unity | 0 referencias
    void Start()
    {
        elScript = pelota.GetComponent<MoveRandomly>();
        esperandoOkHoming = false;
        if (!usarMaquina)
        {
            elScript.setEmpezarMover(true);
        }
    }

    // Update is called once per frame
    Mensaje de Unity | 0 referencias
    void Update()
    {

    }

    0 referencias
    void OnSerialLine(string line)
    {
        if (usarMaquina)
        {
            Debug.Log("Got a line: " + line);
            if (line.Contains("ALARM"))
            {
                elScript.setEmpezarMover(false);
                Debug.Log(line);
            }
            else
            {
                if (!esperandoOkHoming)
                {
                    if (line.Contains("[MSG:'$H'|'$X' to unlock]"))
                    {
                        Debug.Log("La máquina está preparada");
                        esperandoOkHoming = true;
                        Serial.WriteLn("$h");


                    }
                }
                else
                {
                    if (line.Contains("ok"))
                    {
                        esperandoOkHoming = false;
                        elScript.setEmpezarMover(true);
                    }

                }
            }
        }
    }
}

```

Ilustración LXIII Script RecibirLineas

En el código, podemos ver, en primer lugar, la creación de una variable booleana que indica si se va a usar o no la máquina. De esta manera, podemos ejecutar la simulación sin mandar o recibir mensajes de la máquina, es decir, sin que se mueva el proyecto real.

Si estamos utilizando la máquina, comprobamos si estamos en estado de alarma. En caso afirmativo, asignamos a false una variable del Script MoveRandomly que se encarga de indicar si la mascota simulada puede moverse o no. Por otro lado, tampoco queremos

que se mueva hasta que la máquina no haya realizado el homing, por lo que, al empezar, tiene valor false.

Como hemos explicado, tenemos que mandar la instrucción de homing cuando se recibe la línea [MSG:'$H'|'$X' to unlock]. Cuando esto sucede, además de mandarle a la máquina la instrucción de homing, ponemos el valor de esperando homing a true. Esta variable, la utilizamos para indicar que estamos esperando a recibir el ok para confirmar que se ha acabado el homing, si no la tuviésemos, mandaría continuamente hacer homing hasta recibir el ok (se guardaría cada una de las veces en el buffer de la máquina, por lo que tendría que repetir el homing todas ellas para vaciarlo). Una vez acaba el homing, indicamos al Script encargado del movimiento de la mascota que esta ya se puede empezar a mover.

```
//mandar a la impresora la posición
if (usarMaquina)
{
    if (tiempoUpdate >= limiteTiempoMAquina)
    {
        //si no estoy en la cama
        if(destinoCama.x != this.transform.position.x || destinoCama.z != this.transform.position.z)
        {
            string mensajemandar = ("G1 x" + this.transform.position.z * 10 + " y" + this.transform.position.x * -10 + " f" + velocidadMandar).Replace(",", ".");
            Debug.Log(mensajemandar);
            Serial.WriteLn(mensajemandar);
            estoyCama = false;
        }else if (!estoyCama)
        {
            Debug.Log("estoy cama");
            string mensajemandar = ("G1 x" + this.transform.position.z * 10 + " y" + this.transform.position.x * -10 + " f" + velocidadMandar).Replace(",", ".");
            estoyCama = true;
        }

        tiempoUpdate = 0.0f;
    }
    tiempoUpdate += Time.deltaTime;
}
```

Ilustración LXIV instrucción de movimiento de la máquina

Desde el Script movimiento, comprobamos si vamos a usar la máquina. En caso afirmativo, y si el tiempo que ha pasado desde la última vez que mandamos una instrucción al controlador es mayor al requerido, enviamos una nueva instrucción de movimiento.

Esta instrucción, recoge la posición en la que se encuentra la mascota simulada dentro de la escena y se la manda al controlador multiplicada por 10. De esta forma, como el proyecto está escalado, se envía la posición a la que tienen que ir los ejes.

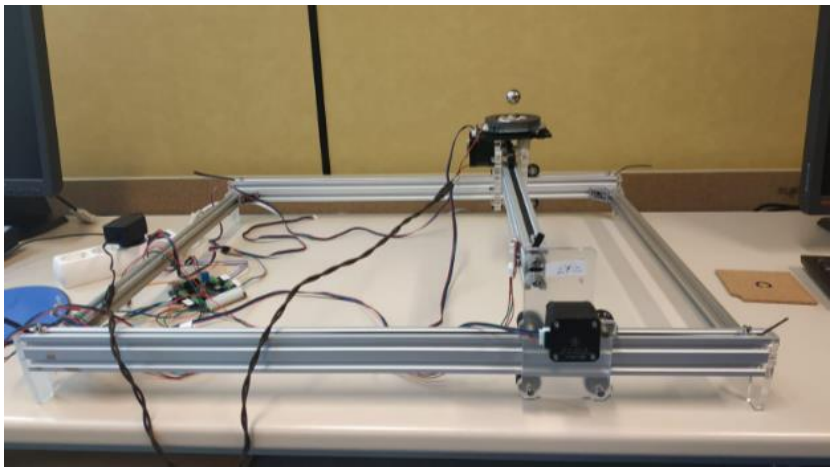

Ilustración LXV Ejes con levitador

CONTRIBUCIONES

Con este proyecto, hemos creado un software que es capaz de comunicar los diferentes módulos necesarios para la realización de una mascota robótica con la particularidad única de la levitación.

Para ello, hemos realizado un script de detección para una pelota, un simulador 3D y la comunicación entre cada uno de los apartados. Además, hemos utilizado una librería ya implementada para la detección de emociones. Por último, hemos construido e incorporado todos los elementos necesarios del hardware.

CONCLUSIONES

En cuanto a lo personal, he disfrutado mucho haciendo este proyecto. No obstante, ha tenido sus partes buenas y sus partes malas.

Empezando por lo positivo, una de mis partes favoritas ha sido la posibilidad de aprender sobre diversos campos que no controlaba, como la levitación magnética, estañar, G-code y todo el hardware en general. Además, como el proyecto admitía todo tipo de nuevas interacciones, he podido entrenar la imaginación tanto para pensar nuevas funcionalidades, como a la hora de idear diferentes maneras de resolver las dificultades de las mismas.

En cuanto a lo relacionado con el software, he aprendido mucho sobre cómo programar de manera modular y comunicar los módulos unos con otros incluso estando en distintos lenguajes de programación. Sé que este es un conocimiento que voy a utilizar en proyectos futuros, así que estoy contenta de haberlo adquirido.

Por otro lado, nunca había tenido la necesidad de realizar un simulador para algo, por lo que no sabía los beneficios que podía traer ni, de forma clara, de qué manera aprovecharlo para un trabajo que iba a acabar siendo físico y no digital. Trabajando en este proyecto he podido averiguar de primera mano lo útil que es y lo mucho que ha simplificado las tareas que en un primer momento parecían imposibles.

Pasando a la parte menos positiva, a pesar de que, como he comentado, lo que más me ha gustado ha sido poder aprender de hardware y circuitos, al no tener experiencia en este campo, en ocasiones era muy frustrante. A fin de cuentas, a lo que estoy acostumbrada es a que si algo no va en el software, se edita y se acaba el problema. En el caso del hardware, esto no era así, ya fuese porque había cortado algo que no había que cortar y entonces era necesario pedir nuevos componentes, porque un cable no era lo suficientemente bueno y empezaba a sacar humo mi proyecto, porque un interruptor se marcaba como activado cuando no lo estaba y no había una solución evidente para ello o porque al activar los ejes la base del levitador rozaba con la tapa y arrancaba el trabajo de dos días.

En resumen, la parte hardware fue mucho menos sencilla de lo que había anticipado y ha consumido la mayor parte del tiempo del proyecto. A pesar de todo ello, me ha encantado y me gustaría aprender mucho más sobre este tema. Por lo que el balance final del trabajo es muy positivo.

LÍNEAS FUTURAS

Con respecto al futuro de este proyecto, sería interesante que se pudiera conseguir que la mascota levitara más alto. Con esto se podría añadir un eje z para la altura y nuevas formas de expresar emociones mediante el movimiento, por ejemplo, tratando de simular la respiración.

También creo que podría mejorar si tuviese un registro de cada usuario y, si éste le ha tratado anteriormente con agresividad, que la mascota no le salude; por el contrario, si siempre juega con él, que se acerque contento.

Otro de los objetivos podría ser que dibujara en un "jardín zen", es decir, poner arena en el playground y que al pasar por encima dibujara algo.

Una de las partes que podría mejorar es la de detección de emociones. La librería utilizada no ha dado los resultados esperados a pesar de ser la mejor de las probadas y con ese cambio, la interacción con la mascota sería mucho más agradable.

En general, creo que el proyecto se presta a poder añadirle una gran cantidad de diferentes funcionalidades, desde pequeños juegos con el usuario hasta detección de voz para el reconocimiento de órdenes.

REFERENCIAS

[1]. Domotizados. (2018, February 18). ¿Existen Las mascotas artificiales? Domotizados.co. https://domotizados.co/existen-mascotas-artificiales/

[2]. Gonzlez, A. (2021, February 17). Los años 90: Historia, datos y curiosidades del tamagotchi. Vandal Random. https://vandal.elespanol.com/reportaje/random-los-anos-90-historia-datos-y-curiosidades-del-tamagotchi.

[3]. Helgren, C. (2017, October 12). *El Tamagochi, La mascota Virtual de los 90, Está DE Regreso en su 20 aniversario*. RT en Español. https://actualidad.rt.com/actualidad/252603-tamagochi-mascota-virtual-regreso.

[4]. Lapedriza, Á. (2019, April 1). ¿Qué ES la COMPUTACIÓN AFECTIVA? Tecnología++. https://informatica.blogs.uoc.edu/computacion-afectiva/.

[5]. Picard, R. (1997). (rep.). *Affective Computing* (pp. 1–16). Cambridge. http://www.media.mit.edu/~picard/

[6]. Martín Rodríguez, E. (2005, July 9). Levitación Magnética por el Prof. Dr. D. Ernesto Martín Rodríguez, académico numerario por el Prof. Dr. D. Ernesto Martín Rodríguez, académico honorario [web log]. https://www.um.es/acc/levitacion-magnetica-por-el-prof-dr-d-ernesto-martin-rodriguez-academico-numerario/.

[7]. K&J Magnetics, Inc. (n.d.). *Levitación electromagnética*. Electromagnetic Levitation. https://www.kjmagnetics.com/blog.asp?p=electromagnetic-levitation

[8]. Dejan, Jim Green May 7, Dzhurabek Irkaev October 6, Byron Lee January 23, Dejan January 31, Alan January 23, Norman January 24 (2021, February 5). G-code explained: List of most important g-code commands. HowToMechatronics. https://howtomechatronics.com/tutorials/g-code-explained-list-of-most-important-g-code-commands/.

[9]. Grbl. (n.d.). Configuring Grbl V0.9 · Grbl/grbl wiki. GitHub. https://github.com/grbl/grbl/wiki/Configuring-Grbl-v0.9.

[10]. MasterD. (n.d.). Qué ES UNITY y PARA qué sirve. https://www.masterd.es/blog/que-es-unity-3d-tutorial/

[11]. Candil, D. (2014, February 21). Unity, El motor DE desarrollo capaz de Partir la Historia de Los videojuegos en dos. Vida Extra - Consolas y videojuegos: Playstation, XBox, Nintendo, PC. https://www.vidaextra.com/industria/unity-el-motor-de-desarrollo-capaz-de-partir-la-historia-de-los-videojuegos-en-dos

[12]. Pinedo, E. (2020, July 28). *Se ACABÓ la ESPERA: 'CUPHEAD' por fin llega a LA PlayStation 4*. Hipertextual. https://hipertextual.com/2020/07/cuphead-playstation-4.

[13]. About. OpenCV. (2020, November 4). https://opencv.org/about/.

[14]. Introduction. OpenCV. (n.d.). https://docs.opencv.org/master/d1/dfb/intro.html

[15]. Hollander, J. (n.d.). Easy laser cut case design. MakerCase. https://es.makercase.com/#/

[16]. Laser cut bird nest box free vector cdr download. 3axis.co. (n.d.). https://3axis.co/laser-cut-bird-nest-box-cdr-file/p7ymw927/.

[17]. Fundaci, & March, F. J. (n.d.). Escala PENTATÓNICA – Músicas no escritas: El PODER de LA IMPROVISACIÓN – Guías DIDÁCTICAS – Recitales para JÓVENES – Música • Fundación JUAN March. Fundación Juan March. https://www2.march.es/musica/jovenes/musicas_no_escritas/pentatonica.asp .

[18]. brusspup. (2020, December 16). *6 amazing MAGNET GADGETS!* YouTube. https://www.youtube.com/watch?app=desktop&v=0eL42AGl2hk

[19]. Tubo de Lenz. experimentos. (n.d.). https://experimentosfisicauc.wixsite.com/experimentos/tubo-de-lenz

[20]. Solenoide 12V efecto LINEAL 3.7x1.5mm Jf0530b. Moviltronics. (2021, April 26). https://moviltronics.com/tienda/solenoide-12v-efecto-lineal-37x15mm-jf0530b/.

[21]. Banggood.com. (n.d.). Escritorio de 500 mW DIY LÁSER Grabadora máquina de GRABADO DE madera MARCADO de IMAGEN impresora CNC 65X50 CM. www.banggood.com. https://es.banggood.com/500mW-Desktop-DIY-Laser-Engraver-Wood-Engraving-Machine-Picture-Marking-CNC-Printer-65x50CM-p-1718021.html?rmmds=search&cur_warehouse=CN.

[22]. Final de Carrera Mecánico CNC Ramps 1.4 IMPRESORA 3D. solectroshop.com. (n.d.). https://solectroshop.com/es/drivers-y-kits-de-control/415-final-de-carrera-mecanico-cnc-ramps14-impresora-3d.html?gclid=CjwKCAjwmqKJBhAWEiwAMvGt6AwDJxMAFYWx4eI41LS90liMMOegBw5cIJY2GQtTyjktHi-nF7eyShoCDYMQAvD_BwE.

[23]. *Cinta Adhesiva de lámina De COBRE DE Dos lados DE 10mm x 25M PARA Blindaje EMI: CINTA: - AliExpress*. aliexpress.com. (n.d.). https://es.aliexpress.com/item/4000750051047.html.

[24]. 50Pcs silicone jumper wire (26awg, high temperature resistant). Smart Prototyping. (n.d.). https://www.smart-prototyping.com/50pcs-Silicone-Jumper-Wire-26AWG-High-Temperature-Resistant.

[25]. Sigalkin, V. (2021, July 26). Iam1337/extOSC: EXTOSC is a tool dedicated to simplify creation of applications in Unity with OSC protocol usage. GitHub. https://github.com/Iam1337/extOSC.

[26]. Rosebrock, A. (2021, April 17). *Ball tracking with opencv*. PyImageSearch. https://www.pyimagesearch.com/2015/09/14/ball-tracking-with-opencv/.

[27]. Sudan, B., & Şahin, İ. (2020, December 12). *berksudan/Real-time-Emotion-Detection: 8 emotions detected in real-time with ~77% ACCURACY. USED: Opencv, Python 3, Keras, data preprocessing, deep learning & machine learning techniques.* GitHub. https://github.com/berksudan/Real-time-Emotion-Detection.

[28]. GameDevTraum. (2021, June 21). *{ rigidbody en unity } ¿qué es, para qué sirve y cómo usarlo?* GameDevTraum. https://gamedevtraum.com/es/desarrollo-de-videojuegos/tutoriales-y-soluciones-unity/serie-fundamental-unity/que-es-un-rigidbody-en-unity/.

[29]. French, J. (2021, March 26). *How to convert the mouse position to world space in Unity (2d + 3D)*. Game Dev Beginner. https://gamedevbeginner.com/how-to-convert-the-mouse-position-to-world-space-in-unity-2d-3d/.

[30]. Prossel. (2018, April 13). *prossel/UnitySerialPort: Script to work with serial port in Unity. optimized to work with Tsv, CSV or other character Delimited values*. GitHub. https://github.com/prossel/UnitySerialPort.

REFERENCIAS ILUSTRACIONES

ANEXOS

```
$0=10
$1=25
$2=0
$3=0
$4=0
$5=0
$6=0
$10=19
$11=0.010
$12=0.002
$13=0
$20=1
$21=1
$22=1
$23=0
$24=25.000
$25=500.000
$26=250
$27=2.000
$30=1000
$31=0
$32=0
$100=250.000
$101=250.000
$102=250.000
$110=3000.000
$111=3000.000
$112=3000.000
$120=250.000
$121=150.000
$122=150.000
$130=200.000
$131=200.000
$132=200.000
```

Ilustración LXVI configuaración grbl