



Universidad
Zaragoza



Proyecto Fin de Carrera

Ingeniería Industrial

Brazo robotizado controlado mediante sensores de visión integrado sobre un robot móvil

Autor:

Pierre Herman

Directores:

Ana Cristina Murillo Arnal

Alejandro Rafael Mosteo Chagoyen

Escuela de Ingeniería y Arquitectura, Universidad de Zaragoza

Febrero 2013

BRAZO ROBOTIZADO CONTROLADO MEDIANTE SENSORES DE VISIÓN INTEGRADO SOBRE UN ROBOT MÓVIL

Resumen

En este proyecto se ha realizado todo el proceso para construir, controlar e integrar un brazo robotizado sobre un robot móvil estándar, en particular la plataforma *TurtleBot*, construido sobre la aspiradora autónoma *Roomba*. Así, el robot adquiere la capacidad para apartar o recoger objetos que pudieran molestarle en el cumplimiento de su tarea de limpieza. La detección e identificación se ha realizado mediante un sensor RGB-d (visión y profundidad), en particular el sensor *Kinect* de Microsoft.

En resumen, se ha realizado un completo desarrollo, tanto software como hardware, de un brazo-robot y los elementos de percepción y control necesarios para su integración en la plataforma robótica móvil existente. El brazo robotizado a construir es de tipo RRR (es decir, con 3 articulaciones de rotación), con una muñeca de un grado de libertad y terminado en una pinza. La detección de objetos y guiado sencillo del brazo se realiza mediante el procesado de información capturada por el sensor RGB-d utilizado (*Kinect*). A partir de la nube de puntos en 3D que se obtiene de este sensor, se detectan objetos que aparezcan a lo largo de la trayectoria del robot y se calcula el punto de agarre donde se dirigirá el brazo.

En más detalle, las tareas realizadas en este proyecto han sido las siguientes:

- A. Diseño y construcción del brazo robotizado:
 - Estudio de alternativas para su construcción, realización de planos 3D, solicitud de presupuestos para las piezas necesarias para construirlo.
 - Cálculo del modelo cinemático para el brazo robotizado.
 - Estudio, y puesta en marcha de un microcontrolador (*Arduino Uno*).
 - Montaje y programación del control del brazo robotizado desde *Arduino*.
- B. Estudio, instalación y familiarización con el entorno *ROS* (pseudo-sistema operativo utilizado para gestión de plataformas robóticas) y sensores relacionados, incluyendo: drivers *OpenNI* para comunicación con sensores *Kinect*; librerías *PCL* y *OpenCV* para facilitar las operaciones con imágenes 3D y 2D respectivamente; librerías para interacción entre *ROS* y *Arduino*
- C. Estudio e implementación del módulo de visión por computador con el sensor *Kinect*. Se han trabajado técnicas de segmentación de planos y agrupación en *clusters* para una detección sencilla de objetos/obstáculos sobre el suelo.
- D. Integración del control del brazo y el módulo de visión sobre el robot móvil:
 - Estudio del funcionamiento del robot-aspiradora y realización de un programa básico de control de movimiento del robot.
 - Implementación del programa principal que transmite la información obtenida por el sistema de visión al robot-aspiradora y al brazo-robot.

Índice

Resumen	1
Índice	2
1. Introducción	4
1.1. Objetivos y Entorno de Trabajo	4
1.2. Distribución temporal	5
1.3. Contenido de la memoria.....	6
2. El brazo robotizado.....	7
2.1. Concepción del robot	7
2.1.1. Fabricación a partir de chapas de aluminio	7
2.1.2. Uso de Kit.....	8
2.1.3. Elección de la alternativa.....	10
2.2. Parte eléctrica.....	10
2.3. Parte de control	12
2.3.1. Plaqueta de micro-controlador	12
2.3.2. La interfaz de programación	13
2.3.3. Cálculo del modelo inverso.....	13
2.4. Diferencias entre modelo y realidad.....	14
2.4.1. Calibración de los servos	14
2.4.2. Calibración del brazo	16
2.5. Código de control.....	16
3. Sistema de visión	17
3.1. Sensor.....	17
3.2. Interfaz de mando.....	19
3.2.1. Conexión de la Kinect	19
3.2.2. Visualización.....	19
3.3. Algoritmos de segmentación.....	20
3.3.1. Reducción del tamaño de la nube de puntos	20
3.3.2. Detección del plano del suelo	21
3.3.3. Detección de los objetos.....	22
3.3.4. Determinación del objeto de interés.....	22
3.3.5. Transferencia de coordenadas.....	23

4.	Prototipo completo	25
4.1.	El robot móvil <i>TurtleBot</i>	25
4.2.	ROS (<i>Robot Operating System</i>)	25
4.3.	Funcionamiento del conjunto	25
4.4.	Resultados	26
4.4.1.	Prueba del error del brazo	27
4.4.2.	Prueba del error del sistema de visión	29
4.4.3.	Límites de alcance	31
5.	Conclusión	32
5.1.	Conclusiones del trabajo y trabajo futuro	32
5.2.	Conclusión personal	33
	Bibliografía	34
	Tabla de las ilustraciones	36
	Anexos	37
	Anexo I: Calculo del modelo inverso	I
	Anexo II: El antiguo brazo robotizado	V
	Anexo III: Presupuestos	VI
	Anexo IV: Planos de las piezas	XI
	Anexo V: Renderizados del brazo robotizado	XXIII
	Anexo VI: Resultados de la calibración	XXVI
	Anexo VII: Especificaciones técnicas de la Kinect	XXIX
	Anexo VIII: Fabricación de piezas “artesanales”	XXX
	Anexo IX: Manual de Usuario del sistema	XXXIII

1. Introducción

Desde pequeño sueño con los robots. Probablemente los dibujos animados, Hollywood y los legos tienen algo que ver. Pero este sueño se ha quedado grabado en mi mente y, mientras los años pasaban, orientaba mi carrera en consecuencia. Es el motivo de mi presencia en esta doble titulación en el CPS, con la especialidad “Automatización y Robótica”. Mi proyecto de fin de carrera está en continuidad directa con la orientación de mi carrera.

La presencia de la tecnología ha aumentado considerablemente en nuestras vidas durante las últimas décadas. La evolución de la robótica de servicio es también notable. Hoy en día, es bastante común encontrar robots de servicio en entornos interiores. Por ejemplo, desde la aparición del Roomba [1], el uso de robots-aspiradora autónomos ha aumentado considerablemente.

La aparición de elementos programables *low cost* y *open-source*, como la plataforma Arduino [2] [3] o la Kinect [4], favorece las aplicaciones en sistemas autónomos personalizados [5]. Permiten la elaboración de robots caseros [6] guiados por sensores que captan información para reconocer elementos del entorno [7].

El *Robot Operating System* (ROS) [8], un *meta-operating system* para robots, permite la interacción y la comunicación entre los diferentes elementos programables de un robot, favoreciendo el desarrollo de robots de servicio completos.

1.1. Objetivos y Entorno de Trabajo

El objetivo final de este proyecto es construir y controlar un brazo robotizado e integrarlo en un robot-aspiradora del laboratorio de robótica. Se basa en la anterior construcción de un brazo hecho como *hobby* en mi tiempo libre, a partir de chapas de madera reciclada, servo-motores y una plaqueta de microcontrolador *Arduino Uno*. El trabajo se ha realizado en el laboratorio de robótica del grupo *RoPeRT*, en las instalaciones del I3A.

Los requisitos del sistema final son:

- El robot tiene que detectar e identificar los objetos que pudieran molestarle en el cumplimiento de su tarea de limpieza mediante un sensor RGB-d (sensores de visión y de profundidad) como por ejemplo el sensor *Kinect* de *Microsoft*.
- Una vez detectado el objeto, y si corresponde a un objeto que se puede recoger y que está situado en la zona de alcanzabilidad del brazo robotizado, el robot deberá actuar para recogerlo permitiéndole continuar el cumplimiento de su tarea de limpieza.

Las tareas más importantes a realizar han sido las siguientes:

- La **construcción del brazo robotizado**. Esto necesita un estudio de las posibles alternativas, a saber: o construir totalmente el brazo a partir de chapas de aluminio, o usar un kit de montaje que permite un ensamblaje sencillo. Dado el caso de la fabricación completa del brazo robotizado, necesitamos pedir presupuesto a diversas empresas y planos de dicho brazo, que será modelado con CatiaV5 (programa de CAD). El brazo robotizado es de tipo RRR (es decir con 3 articulaciones de rotación) con una muñeca de un grado de libertad y terminado en una garra. La estructura externa está construida con piezas de aluminio, los actuadores son servo-motores y el conjunto está controlado mediante una plaqueta de micro-controlador, la plaqueta Arduino Uno [9].
- El cálculo del **modelo cinemático inverso** del brazo [10] permite obtener todas las configuraciones posibles de nuestro robot (en este caso solo son dos configuraciones porque la muñeca no tiene suficientes grados de libertad para permitir más). Así, con el modelo inverso, dada una posición (las coordenadas x , y , z) y el ángulo entre la garra y el suelo, obtenemos la configuración necesaria del robot para alcanzar dicha posición, es decir los ángulos donde debemos colocar cada eslabón.
- Diseño e implementación del **sistema de visión**. Este proceso utilizará las nubes de puntos captadas por el sensor RGB-d de la Kinect, que contienen la información específica de cada punto captado, como las coordenadas o el color. A partir de esta información, mediante la biblioteca de PCL (*point-cloud library*) [11] y ROS [8], vamos a segmentar los puntos con diferentes modelos para permitir la detección del objeto de interés y su centroide. Finalmente el sistema debe elegir el objeto más cercano al robot, para recogerlo.
- Implementar un **módulo sencillo de movimiento**, para realizar desplazamiento del robot de base (*Roomba*) y probar el funcionamiento del sistema completo.
- **Integrar y coordinar todos los módulos** (control del brazo, sistema de visión, módulo de desplazamiento). Para ello se utilizara ROS, mediante el sistema de *publish & subscribe*. Es decir, cada programa envía o recibe mensajes de otros programas. Además, ROS tiene la ventaja de poseer varios drivers como los drivers de la Kinect y del Roomba, pero también dispone de un programa de visualización que es muy útil para verificar el funcionamiento del código de visión.

1.2. Distribución temporal

En la ilustración siguiente (Ilustración 1), se descomponen los tiempos por semana de cada actividad.

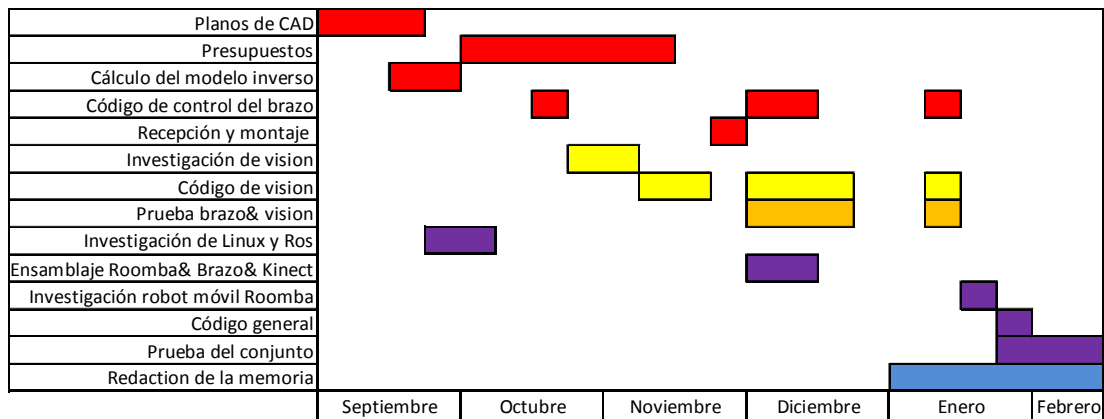


Ilustración 1: Diagrama temporal del trabajo

NB: en rojo vemos los trabajos propios del brazo. En amarillo, los trabajos de visión. En naranja, los de visión y del brazo integrados. En Violeta, las actividades que conciernen al conjunto, es decir el sistema final. En azul, la redacción de la documentación.

1.3. Contenido de la memoria

El primer capítulo trata del desarrollo del brazo robotizado, desde su concepción hasta su control. Se explican las diferentes alternativas posibles para el montaje de la estructura externa, así como la alternativa elegida. Además, hablaremos del funcionamiento, de su control y de los aspectos propios de la concepción de un modelo real.

A continuación, el segundo capítulo desarrolla la parte “visión” del trabajo. Empieza por una breve explicación del funcionamiento del sensor RGB-d utilizado, es decir la cámara Kinect de Microsoft. Hablaremos de las interfaces de este sensor, para acabar con los diferentes algoritmos de segmentación que permiten la detección de objetos.

Después, el capítulo siguiente trata del prototipo completo. Empieza describiendo el robot móvil Roomba, para después desarrollar el funcionamiento del conjunto (brazo robotizado con el sistema de visión y el robot móvil Roomba). Terminaremos con un análisis breve de los resultados obtenidos.

Para acabar, concluiremos sobre el proyecto y las maneras de mejorarlo.

2. El brazo robotizado

Esta parte del proyecto se basa en un brazo robotizado que hice anteriormente, en mi tiempo libre, a partir de chapas de madera reciclada, servo-motores y una placa de microcontrolador Arduino Uno (ver Anexo II: El antiguo brazo robotizado). Su poca precisión y su falta de actuador para el agarre de objetos implican una reforma total de la estructura. El aluminio parece el material más adaptado a este caso por sus propiedades. Además, la parte de programación del antiguo brazo solo implementaba un modelo geométrico sencillo para el cálculo de los ángulos mientras que ahora queremos utilizar el modelo geométrico inverso que permite obtener todas las posiciones alcanzables (ver Anexo I: Calculo del modelo inverso).

2.1. Concepción del robot

Las dos alternativas posibles estudiadas para la realización de la estructura externa del brazo robotizado son:

- La fabricación de las distintas piezas a partir de chapas de aluminio, que permite construir un brazo más adecuado al uso deseado.
- El uso de un kit que permite estabilizar los servomotores y asegurar la rotación entre ellos. Además el kit es compatible con los servomotores estándares.

En ambos casos queremos un robot con cuatro grados de libertad y una garra. Los actuadores serán servomotores y el control será efectuado por una plaqueta de microcontrolador Arduino.

2.1.1. Fabricación a partir de chapas de aluminio

La idea de esta alternativa es realizar la estructura externa a partir de chapa de aluminio, limitando las operaciones de fabricación como el corte y el pliegue. Las ventajas siguientes hacen del aluminio el componente ideal para la realización de nuestro brazo robotizado:

- Material ligero.
- Resistente.
- Fácil de cortar y plegar.
- Relativamente barato.

Para reducir las fases de fabricación, utilizamos piezas estándares como los tirantes (de 25mm y 50mm) y los tornillos (M3). La estructura está hecha para pesar el mínimo posible, guardando una buena rigidez y también para funcionar con los servomotores ya poseídos (3 Hitec HS-311, un Hitec HS-645MG y un mini-servo estándar).

El programa de CAD CatiaV5 no solamente permite diseñar los planos 3D (ver Anexo V: Renderizados del brazo robotizado) del brazo robotizado, sino también calcular los centros

de gravedad (implementando los parámetros de masa por volumen de cada elemento) y las masas equivalentes. Así, podemos verificar que los pares de fuerza en las articulaciones son inferiores a los pares de los servomotores.

Otro aspecto interesante de los programas de CAD es la posibilidad de diseñar automáticamente los planos 2D a partir de las piezas 3D. Una modificación en la concepción 3D se actualiza en los planos 2D (ver Anexo IV: Planos de las piezas y Anexo V: Renderizados del brazo robotizado).

Con estos planos contactamos varias empresas para obtener presupuestos.

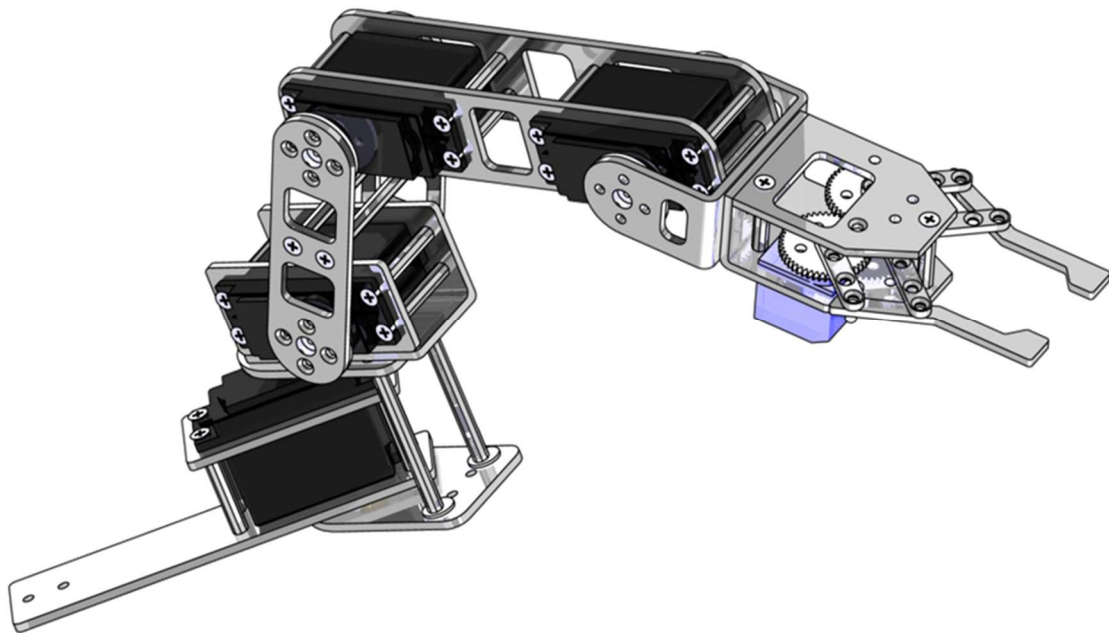


Ilustración 2: Renderizado de la vista 3D de CatiaV5

2.1.2. Uso de Kit

La segunda alternativa es el uso de un kit de montaje adaptable a los servomotores. El kit está compuesto por *bricks* (piezas modulares) que se pueden ensamblar entre ellos de manera muy sencilla [12].

Cada *brick* permite reforzar el eje de rotación gracias a una barra de carga fijada en una jaula de aluminio, alineada con el eje del servomotor. Así, la horquilla y la jaula están en rotación entre ellos de manera rígida, lo que limita los movimientos indeseados en las otras direcciones (ver Ilustración 3).

Las ventajas del uso de un kit son:

- Adaptabilidad a los servomotores.
- Sencillez de montaje.

- No necesitamos operaciones de fabricación.
- Seguridad de tener un sistema que encaja con precisión.

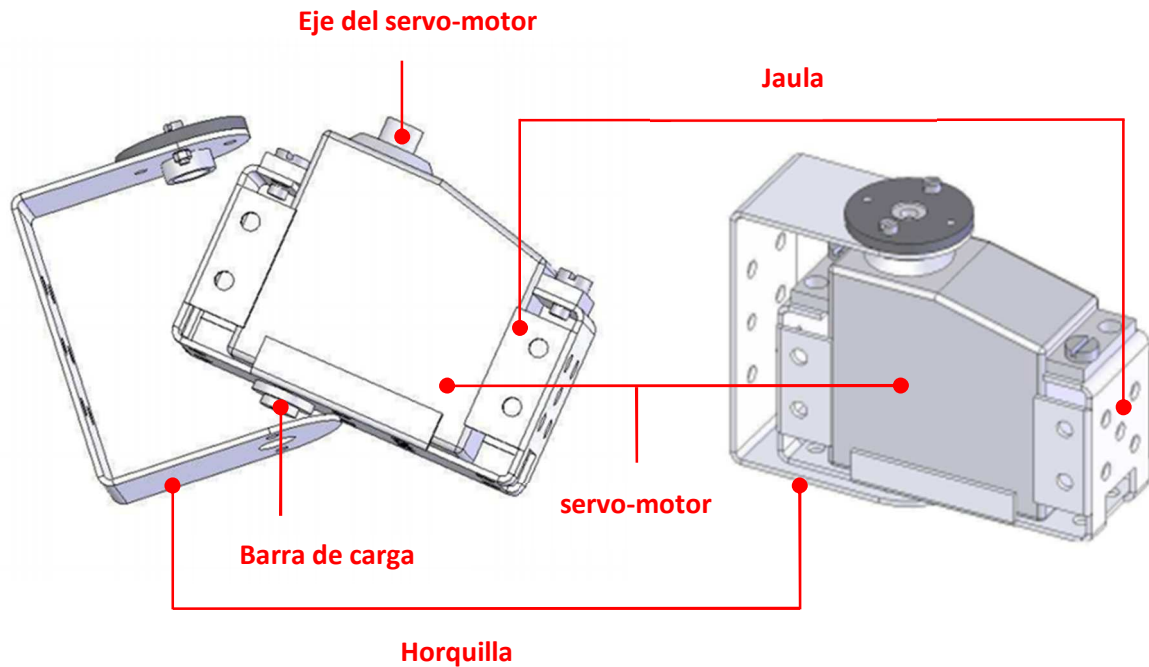


Ilustración 3: Descomposición de *brick* EasyRobotic

Un punto negativo del kit es que la garra solo es compatible con el servomotor Futaba S3003, que pesa más que el mini servo. En consecuencia, a la lista de las piezas tenemos que añadir un servo Futaba S3003 y un servo con más par como, por ejemplo, el HBKing (9kg/cm).



Ilustración 4: Ejemplo de robot construido con kits EasyRobotic

La Ilustración 4: Ejemplo de robot construido con kits EasyRobotic [12], nos muestra que el kit permite la creación de varios tipos de robots.

2.1.3. Elección de la alternativa

Los presupuestos de las dos alternativas son más o menos de precio equivalente (Anexo III: Presupuestos). Sin embargo, optamos por la solución del Kit de EasyRobotic, por varias razones.

La primera es que las peticiones de presupuestos y los intercambios con las empresas se alargaron, lo cual añadía a esta opción demasiado tiempo de espera para la recepción de las piezas y el tiempo de fabricación de las empresas.

La segunda es que, con el kit, estamos seguros de que todo va a encajar perfectamente, lo que no es el caso si contratamos con las empresas para construir las piezas a medida del brazo robotizado, lo cual de nuevo sería un retraso muy importante en el proyecto.

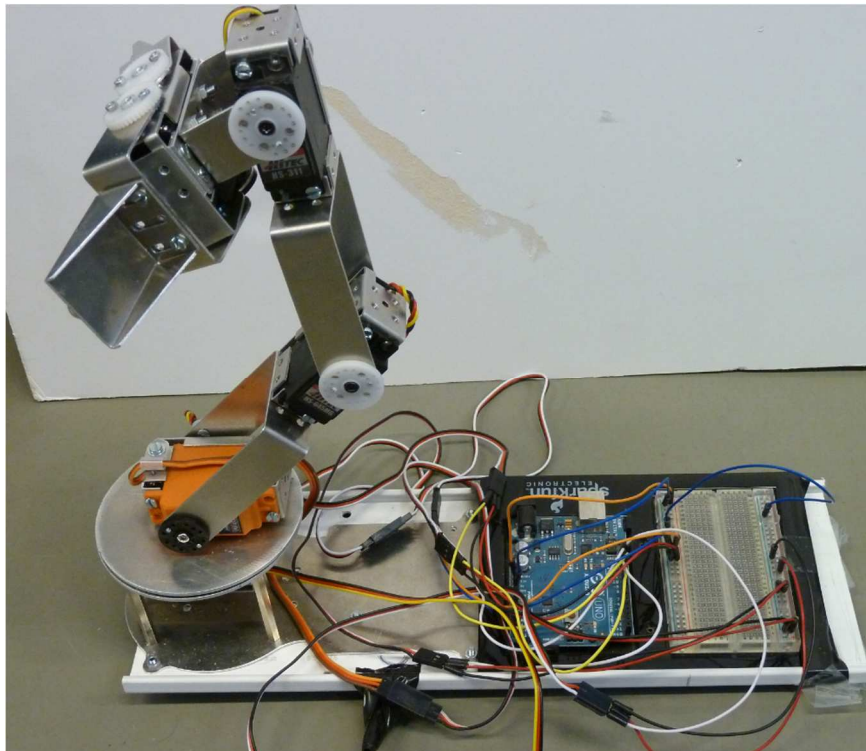


Ilustración 5: Brazo robotizado construido con la alternativa elegida

2.2. Parte eléctrica

Los servomotores [13] permiten desplazar un brazo, que sujeta un objeto, hasta una posición deseada (o ángulo de rotación), y después mantener esta posición. El termino servomotor significa “motor sometido”.



Ilustración 6: Servomotor Hitec HS-311

Un servomotor es un ensamblaje complejo de mecánica y de electrónica que contiene (Ilustración 7):

- un motor de corriente continua de pequeño tamaño
- una carta electrónica de avasallamiento
- un reductor que disminuye la velocidad pero aumenta el par
- un eje que sale del cuerpo que permite acoplar diferentes brazos o ruedas
- un potenciómetro que regula la posición

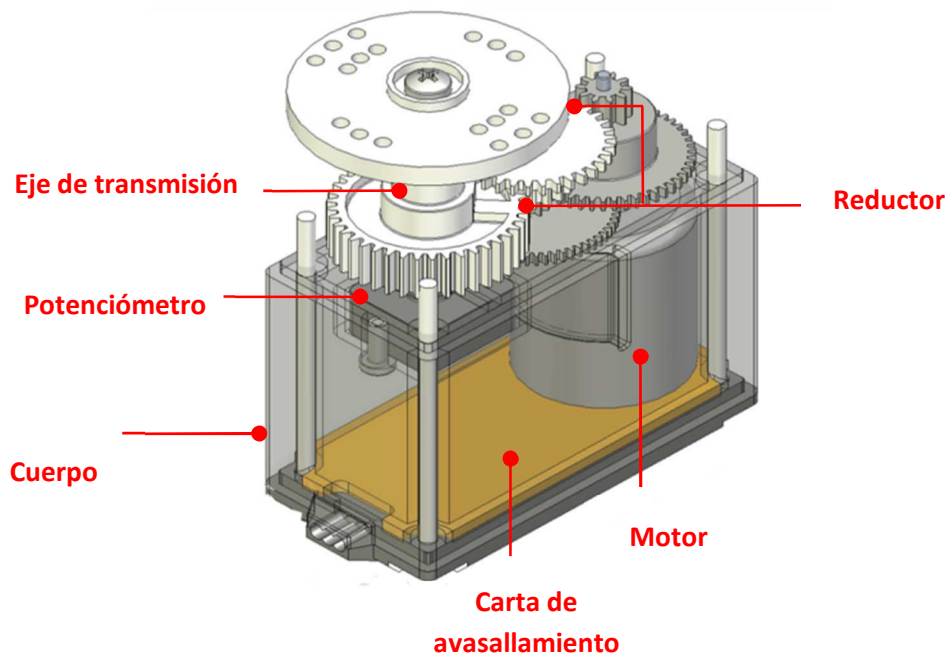


Ilustración 7: Vista del interior del servo-motor

Un servomotor se controla mediante 3 cables eléctricos (Ilustración 6). Estos cables permiten a la vez alimentar (cables positivos y negativos) y transmitir las consignas de posición (por el cable de señal).

Al contrario que un motor de corriente continua, que puede ser controlado por variaciones de tensión o por encendido/apagado, un servomotor reacciona a impulsos de duración variable. La duración de la señal determina la rotación del eje y, por tanto, la posición angular. La impulsión determina la posición absoluta y no relativa. Para mantener la posición, la señal debe ser repetida de forma regular (normalmente cada 20ms)

En nuestro caso los servomotores son los actuadores del robot. De pequeño tamaño y ligeros, están diseñados para el control de juguetes de modelismo. Además, permiten un control de posición relativamente preciso por impulsiones.

Los servos utilizados son de varios tipos y de varias potencias:

- Hitec Hs-311 (par de 3kg/cm, 150 mA, Ilustración 6) x2
- Hitec Hs-645 (par de 9kg/cm, 350 mA)
- Futaba S3003 (par de 3,2kg/cm, 120 mA)
- HBKing (par de 9kg/cm, 340 mA)

2.3. Parte de control

2.3.1. Plaqueta de micro-controlador

La parte de control se hace mediante una plaqueta de micro-controlador de la marca Arduino (la Arduino Uno [9], Ilustración 8). Ésta permite un control sencillo de elementos electrónicos, de manera digital o analógica. Cada conector puede funcionar como una entrada o una salida.

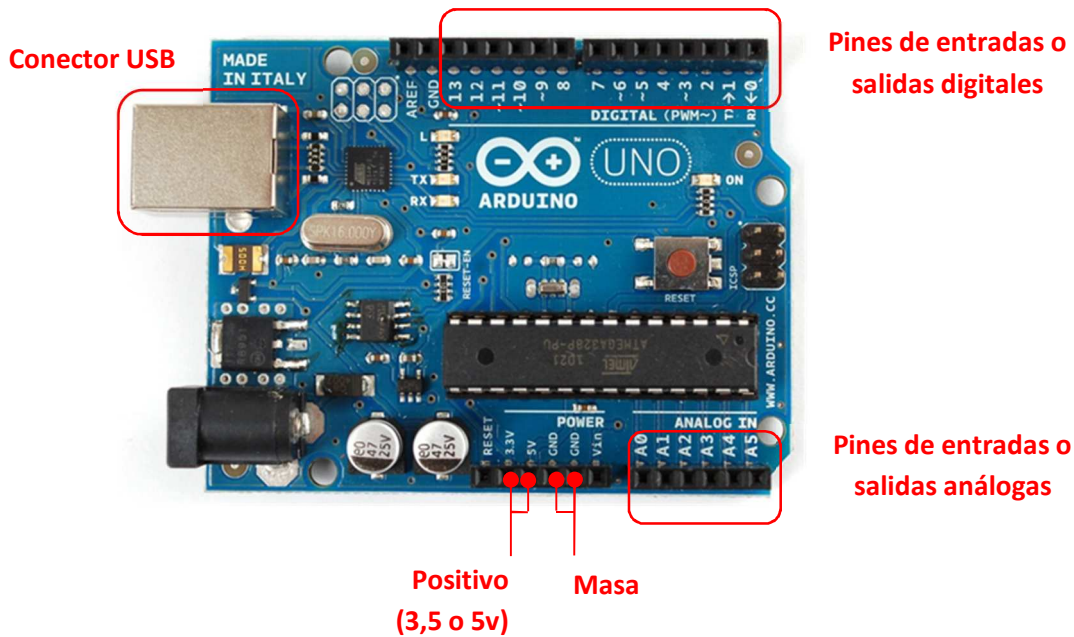


Ilustración 8: Plaqueta de microcontrolador Arduino Uno

El problema de la plaqueta Arduino es que solo puede suministrar 5V y 500mA. El consumo de nuestros servos es de $i = 150 \cdot 2 + 350 + 120 + 340 = 1110 \text{mA}$. Por lo tanto necesitamos una segunda alimentación. Para eso, vamos a utilizar un viejo cable USB que vamos a modificar para solo aprovechar la potencia, descartando los cables de datos (Anexo VIII: Fabricación de piezas “artesanales”).

2.3.2. La interfaz de programación

La conexión entre la plaqueta Arduino [9] y el ordenador se hace mediante un conector USB. El lenguaje de programación utilizado es C, con varias bibliotecas que simplifican el control. Además, el software de interfaz es *Open Source*, lo que favorece su desarrollo colaborativo.

2.3.3. Cálculo del modelo inverso

Utilizando los parámetros DH (Denavit–Hartenberg) del robot, podemos resolver el modelo inverso [10] del brazo robotizado y así obtener las ecuaciones de paso entre las coordenadas y los ángulos de cada servo, y programar las trayectorias [14].

Los caculos están detallados en el anexo correspondiente (Anexo I: Calculo del modelo inverso)

2.4. Diferencias entre modelo y realidad

Con el modelo inverso, tenemos las ecuaciones teóricas de paso entre los ángulos de los servos-motores y la localización espacial del brazo robotizado. Pero ahora tenemos que transferir la teoría a la realidad, con su lote de diferencias.

2.4.1. Calibración de los servos

Los servos están controlados por impulsos. La plaqueta Arduino envía un impulso de duración t , y en función de la duración de este impulso, el servo se mueve a la posición angular α correspondiente. Los fabricantes de los servos aseguran que t y α son proporcionales. Sin embargo, no especifican las correspondencias.

En consecuencia, es necesario calibrar los servos para conocer esta relación de proporcionalidad entre duración del impulso y posición.

- *Parte física del calibrado*

La parte física del sistema de calibración es muy sencilla. Se necesita conocer las diferentes posiciones angulares del servo para varios impulsos. Lo componen un bastidor para recibir el servomotor, un transportador y un peón de medida que permite recoger las medidas (Ilustración 9 y Anexo VIII: Fabricación de piezas “artesanales”).

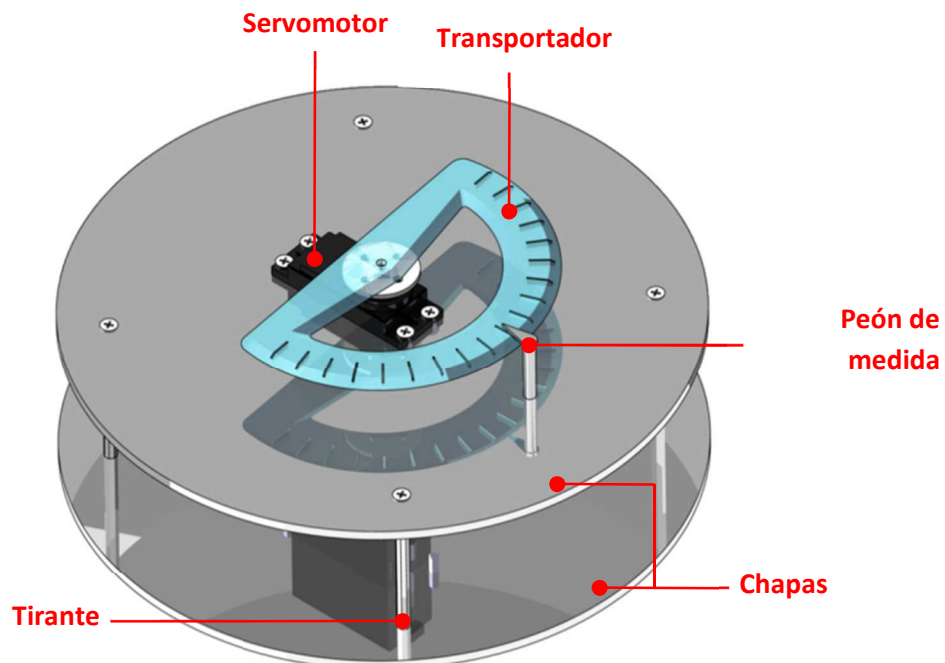


Ilustración 9 : Descripción del calibrador

- *Parte programación*

Para calibrar los servos se necesita una aplicación que permita comunicar directamente el ordenador con la plaqueta. Así, se puede definir un impulso tecleándola, en una terminal de órdenes, y recoger el valor angular correspondiente.

- *Resultados*

Con el programa de calibración anterior, dibujamos con un tabulador, las curvas de conversión entre la impulsión de entrada y el valor angular de salida, y así obtenemos las ecuaciones de paso.

El servomotor Futaba S3003 sirve solo para cerrar o abrir la garra, por lo que no necesitamos conocer la función de conversión entre las impulsiones y los grados.

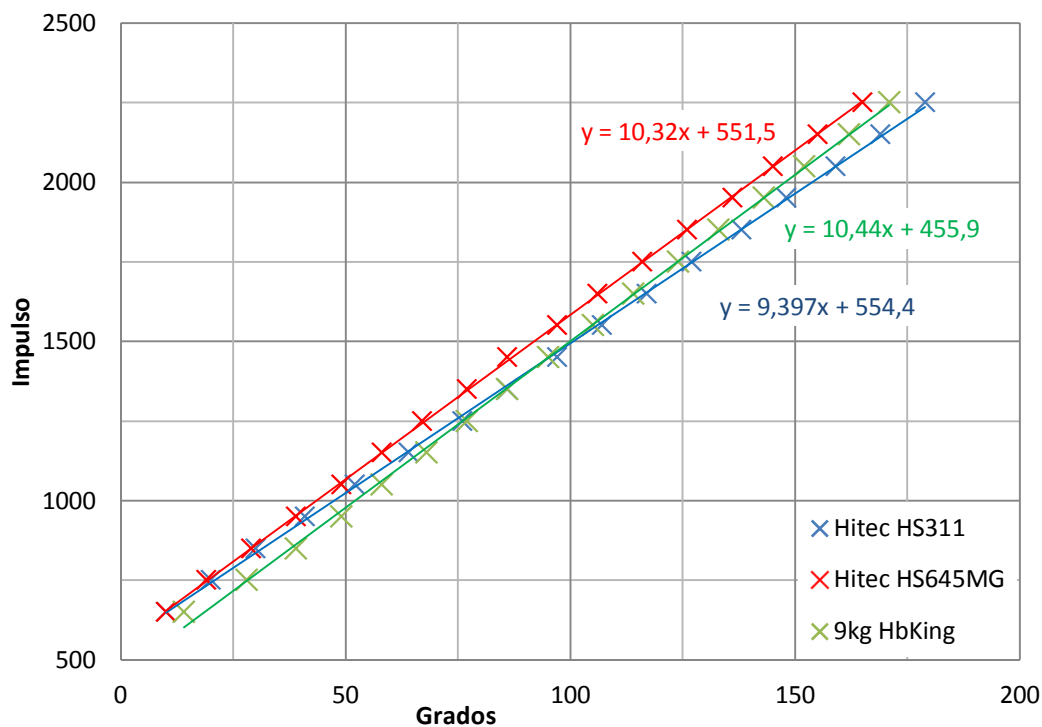


Ilustración 10: Curva de conversión de grados a impulsión

Notamos que estas curvas son lineales (Ilustración 10), como lo precisa la documentación técnica de los servos. Sin embargo, las curvas son suficientemente diferentes como para justificar la implementación de tres funciones de conversión (y no una única para todos los servos).

Más detalles en Anexo VI: Resultados de la calibración.

2.4.2. Calibración del brazo

Otra diferencia con la teoría es que cada servo tiene una gama angular entre 0 y 180 grados. Por lo tanto tenemos que montar el robot para aprovechar al máximo esta gama angular.

Además, nuestros cálculos del modelo inverso están hechos para una posición inicial. Entonces, tenemos que considerar el 0 angular de cada servo a partir de la posición inicial del cálculo.

Como los servomotores que usamos no proporcionan vuelta de información, vamos a calcular estos desfases angulares de forma aproximada: modificamos poco a poco los ángulos impuestos a cada servo, hasta llegar a la posición inicial utilizada para el cálculo del modelo inverso.

2.5. Código de control

Una vez implementadas las ecuaciones de paso entre coordenadas y ángulo, añadimos varias funciones para controlar el brazo. Las más importantes son las siguientes:

- *Init*, inicializa la posición del robot.
- *Calculate*, calcula los ángulos para una posición deseada.
- *Pose*, desplazamiento articular absoluto.
- *Move*, desplazamiento articular con control de velocidad.
- *Moves*, desplazamiento lineal con control de velocidad.
- *Open*, abre la garra.
- *Close*, cierra la garra.

3. Sistema de visión

Para detectar los objetos a manipular por el brazo robotizado, se implementó un módulo de percepción basado en visión descrito a continuación.

3.1. Sensor

El sensor utilizado es un sensor RGB-d, es decir de visión y profundidad, que permite capturar nubes de puntos tridimensionales. Estas nubes contienen la información específica de cada punto captado, como las coordenadas o el color. A partir de esta información vamos a segmentar los puntos con diferentes modelos para permitir la detección del centroide de nuestro objeto, lo que permitirá al robot actuar en consecuencia.

En nuestro caso, disponemos en el laboratorio de la cámara Kinect de Microsoft (ver Ilustración 11) que es famosa por su innovación en el mundo de los videojuegos, porque permite jugar sin mando, gracias a su sistema de reconocimiento de los jugadores.



Ilustración 11 : Descripción de la Kinect

La cámara RGB captura las imágenes de color RGB como una cámara clásica. El sensor de profundidad utiliza un captor CMOS y un emisor de infrarrojos (para más detalles técnicos, ver Anexo VII: Especificaciones técnicas de la Kinect).

El sensor CMOS de infrarrojos permite obtener una imagen que representa la radiación térmica emitida por el objeto observado. Entonces, las informaciones captadas representan un mapeo de las emisiones térmicas, emitidas por el objeto que miramos (pero no nos transmite ninguna información sobre la profundidad ver Ilustración 12).

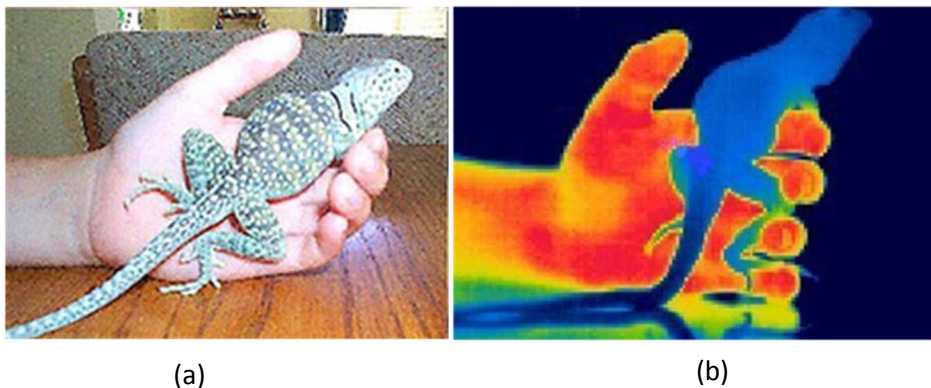


Ilustración 12: imagen RGB (a) e imagen de infrarrojos (b)

El emisor de infrarrojos bombardea su entorno con rayos infrarrojos (ver Ilustración 13). Una parte de estos rayos va a ser reflejada por el conjunto de las superficies tocadas. Así, cuanto más cercano esté un objeto, mayor será la cantidad de radiación reflejada. Por lo tanto, la cámara infrarroja, gracias a estos datos de cantidad de radiación recibida, puede medir la distancia entre la cámara y el objeto.

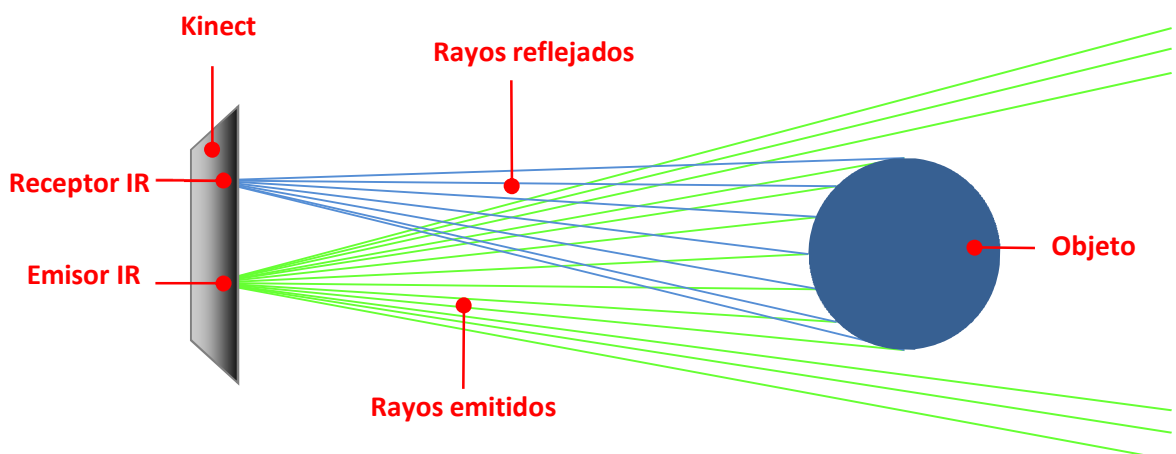
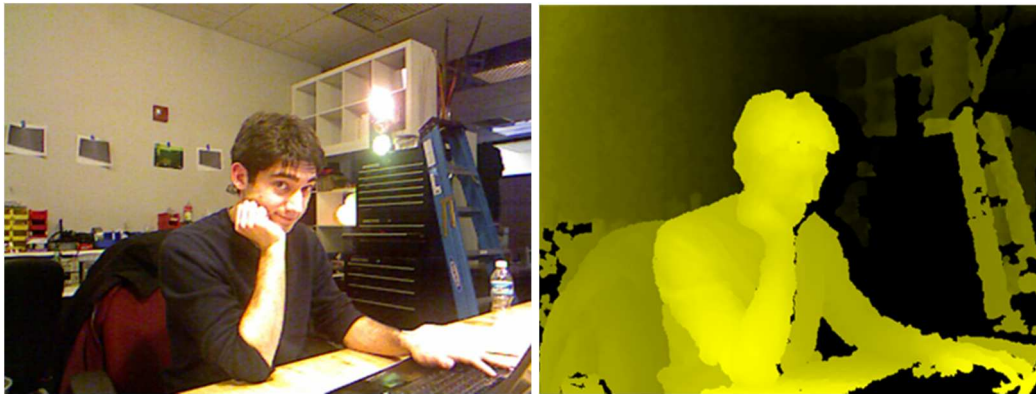


Ilustración 13: Esquema del funcionamiento del detector de profundidad

Con este sensor recuperamos un mapa de profundidad de la escena captada. Cada punto contiene los datos de distancia a la cámara (Ilustración 14), que se guardan de manera más cómoda como coordenadas (x,y,z) en el sistema de coordenadas centrado en el receptor CMOS (Ilustración 20). Estos datos más los datos de la cámara RGB forman las nubes de puntos (*point-cloud* en inglés).



(a)

(b)

Ilustración 14 : imagen RGB (a) e imagen de profundidad (b)

3.2. Interfaz de mando

3.2.1. Conexión de la Kinect

Físicamente la Kinect necesita una alimentación externa y una conexión USB 2.0 para transmitir las nubes de puntos.

La parte software necesita varios drivers que están incluidos en la biblioteca de ROS. Para ponerla en funcionamiento utilizamos la siguiente orden en la terminal:

```
>> roslaunch openni_launch openni.launch
```

3.2.2. Visualización

Para entender mejor el funcionamiento de nuestros algoritmos y también para verificar su buen funcionamiento, vamos a utilizar un programa de visualización 3D que está incluido en la biblioteca de ROS, el programa RVIZ (Ilustración 15).

Lo abrimos directamente en el terminal de Ubuntu con la orden siguiente:

```
>> rosrn rviz rviz
```

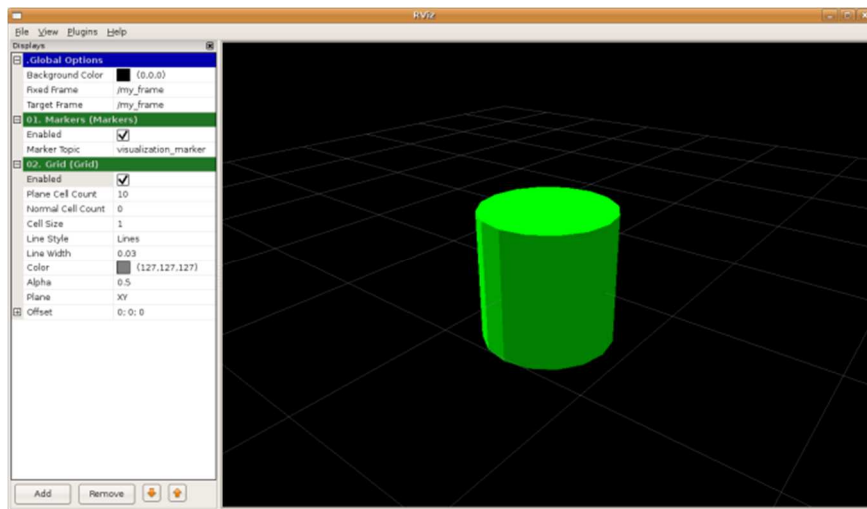


Ilustración 15: Visualizador Rviz

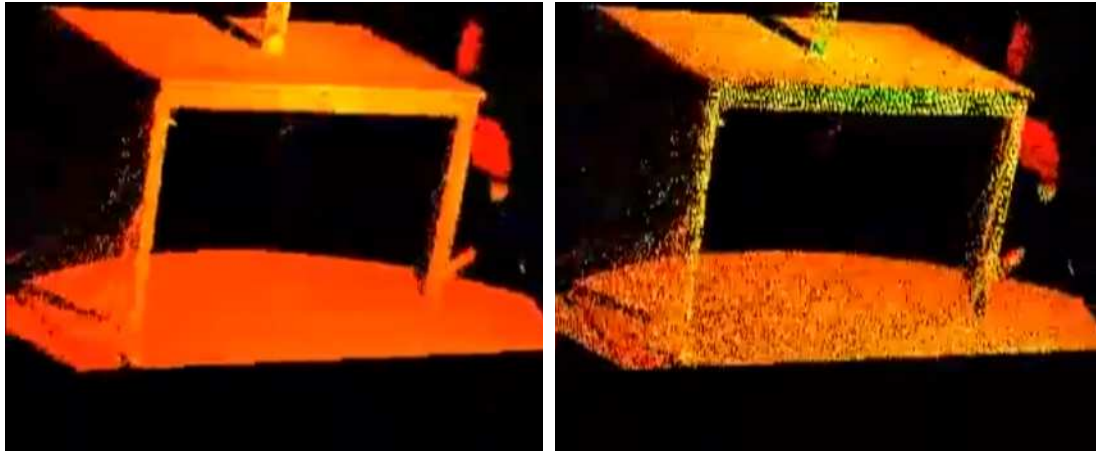
3.3. Algoritmos de segmentación

El objetivo del sistema de visión es detectar los objetos que pueden obstaculizar el funcionamiento de un robot móvil aspiradora en el cumplimiento de su tarea de limpieza, como por ejemplo la ropa desafortunadamente dejada por el suelo.

Además, nos limitamos a los objetos que puede levantar el brazo, es decir ropa de pequeño tamaño como trapos, calcetines, etc.

3.3.1. Reducción del tamaño de la nube de puntos

Las nubes de puntos pueden contener muchos datos, y cuantos más datos tenemos, más largos serán los tiempos de cómputo. Por eso vamos a empezar por una reducción del tamaño de la nube emitida por la Kinect usando el *Voxel-grid* [11]. El *Voxel-grid* cuadrícula el espacio en pequeñas cajas del tamaño deseado (para las 3 dimensiones del espacio), y cada punto englobado en una caja va a representar un único punto. El proceso es análogo a la pixelización en 2D, pero en lugar de representar un “super-pixel” del tamaño de la caja, todos los puntos están aproximados por el centroide.



(a)

(b)

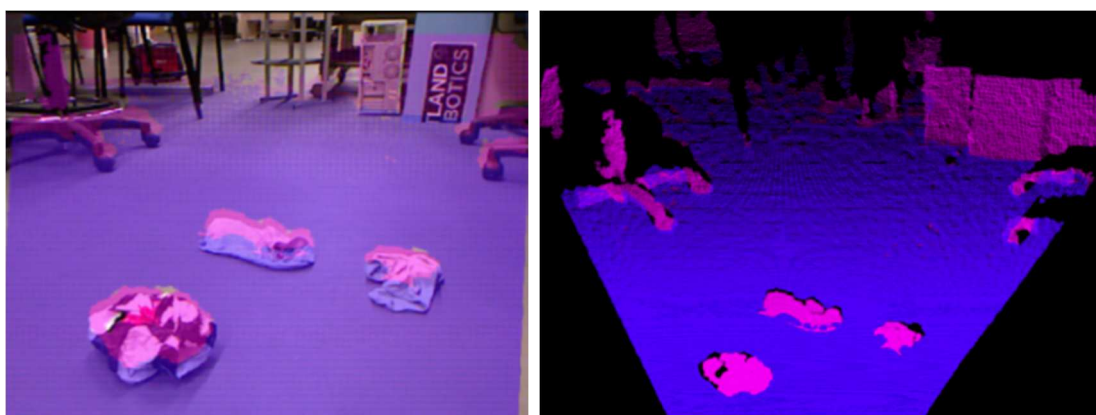
Ilustración 16: (a) Nube de puntos y (b) nube de puntos después del *voxel-grid*.

La Ilustración 16 muestra el resultado de una nube procesada por el filtro *voxel-grid*.

3.3.2. Detección del plano del suelo

En nuestro caso, el robot móvil aspiradora Roomba trabajará únicamente en un entorno interior. Es decir, que solo se desplazará en suelos planos o casi planos. Por lo tanto, podemos usar esta hipótesis para nuestro algoritmo. Es decir, vamos a calcular el plano del suelo para utilizarlo después.

Utilizamos la función “*SACSegmentation*” [11] con el método *RANSAC* [15], que permite obtener una segmentación plana de manera robusta y por lo tanto destacar el plano del suelo de nuestra nube de puntos (Ilustración 17).



(a)

(b)

Ilustración 17: (a) Superposición de la detección con la imagen RGB, (b) reconstrucción en 3D. En azul, el plano del suelo detectado; en rosa, los otros puntos captados sobrantes.

3.3.3. Detección de los objetos

Para detectar los objetos, vamos a utilizar la segmentación por agrupación, utilizando la librería estándar de PCL *Euclidean Cluster Extraction* [11], que implementa el método del K-mean Clustering [16] asociada a una estructura Kd-tree [17]. La idea principal es considerar que si dos puntos no superan una cierta distancia entre ellos, entonces pertenecen al mismo objeto (Ilustración 18). A este parámetro de distancia podemos añadir parámetros como el número mínimo y máximo de puntos para que la agrupación sea considerada como un objeto.

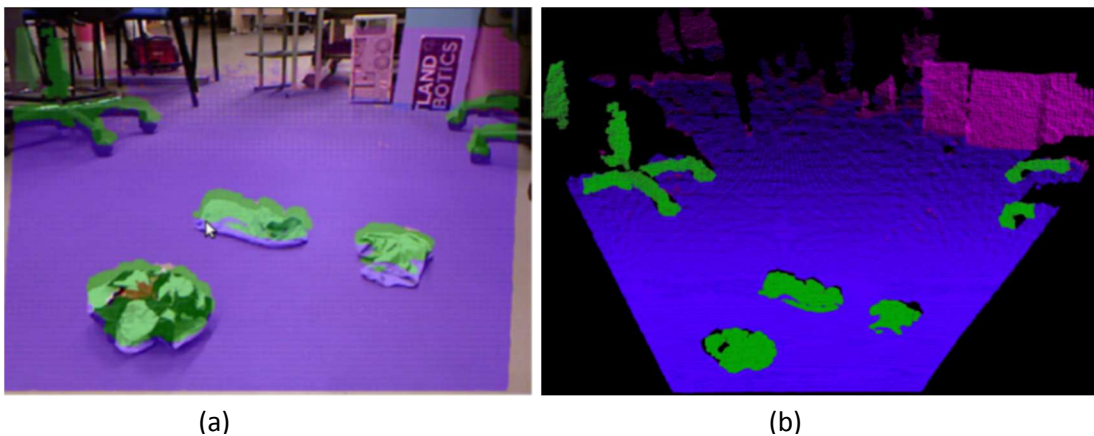


Ilustración 18: (a) Superposición de la detección de objetos con la imagen RGB, (b) reconstrucción en 3D. En verde, los objetos detectados por la agrupación. En azul, el plano del suelo detectado. En rosa, los otros puntos captados sobrantes.

3.3.4. Determinación del objeto de interés

Para determinar el objeto que nos interesa procesar en primer lugar, calculamos el centroide de cada objeto detectado y buscamos cuál de estos objetos tiene el centroide más cercano al robot (Ilustración 19). Tomamos sus coordenadas para enviarlas al programa principal de control del sistema.

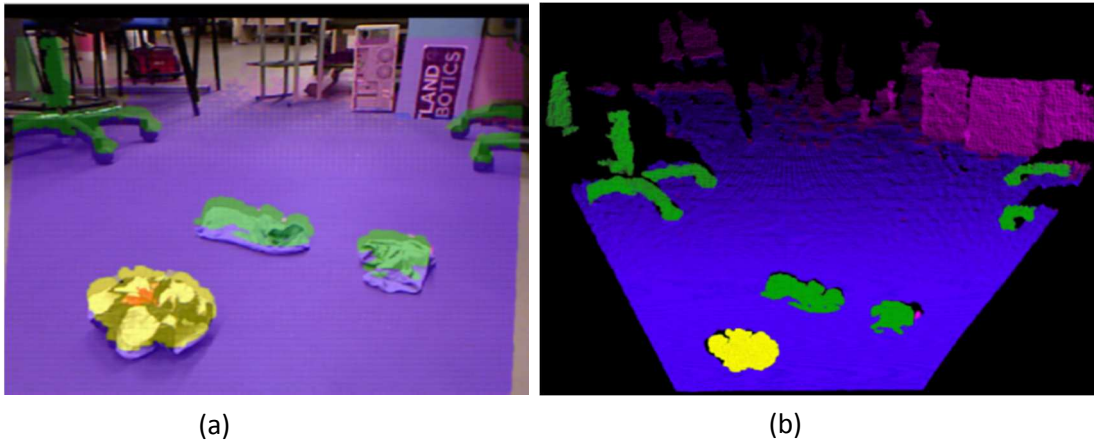


Ilustración 19: (a) Superposición de la detección con la imagen RGB, (b) reconstrucción en 3D. En amarillo, el objeto de interés. En verde, los objetos detectados por la agrupación. En azul, el plano del suelo detectado. En rosa, los otros puntos captados sobrantes.

3.3.5. Transferencia de coordenadas

Ahora que tenemos las coordenadas del centroide de nuestro objeto, tenemos que transformarlas al sistema de la base del brazo robotizado. Entonces, como la cámara esta inclinada hacia el suelo, tenemos que primero hacer una rotación de coordenadas de la base de la cámara (Ilustración 20), para orientarla en la referencia de la base del brazo, es decir en paralelo con el suelo. Para hacerlo, utilizamos los coeficientes del plano calculado con el algoritmo de segmentación plana para calcular el ángulo entre la cámara y el suelo (suponemos que tenemos la cámara alineada para que el eje transversal sea paralelo al suelo). Con este ángulo ya podemos estimar la rotación de la base.

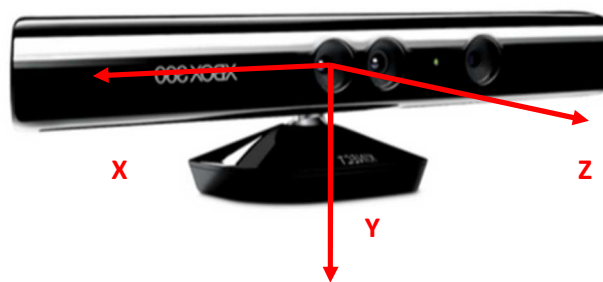


Ilustración 20: Ejes de la Kinect

Después, con las dimensiones propias del robot podemos calcular una traslación de la base rotada a la base del brazo (Ilustración 21).

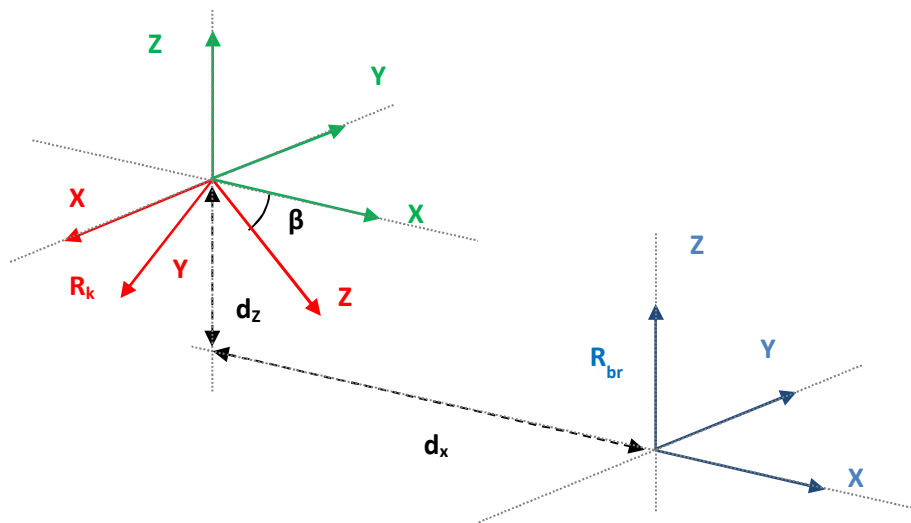


Ilustración 21: Esquema de la transferencia de coordenadas, R_k sistema de coordenadas de la Kinect, y R_{br} de coordenadas del brazo robotizado.

4. Prototipo completo

4.1.El robot móvil *TurtleBot*

El robot móvil es una plataforma estándar comercializada por *Willow Garage* [18] que se programa de manera sencilla, y cuya base móvil es un robot aspiradora comercial *Roomba*. Queremos que el robot haga varios movimientos hasta que encuentre un objeto que recoger. Usando los *drivers* correspondientes, nuestro código solo tiene que publicar datos de velocidad (lineal y angular), y puede suscribirse a los datos obtenidos procesando los sensores de la plataforma móvil (sensor de contacto, de distancia, etc.).

4.2.ROS (*Robot Operating System*)

En este proyecto tenemos varios elementos que tienen que comunicarse entre ellos. Para hacerlo, utilizamos ROS que es un pseudo-sistema operativo, que permite la comunicación entre varios módulos gracias a un sistema de *publish & subscribe* (publicar y suscribir). Además, contiene varias librerías que permiten utilizar *hardware* variado (como la *Kinect* o el *Roomba*).

Otra ventaja de ROS es que incluye un programa de visualización que se revela muy útil para la verificación de nuestro código de visión. Así podemos monitorizar el comportamiento del programa.

4.3.Funcionamiento del conjunto

Cada bloque anteriormente detallado se comunica gracias al pseudo-sistema operativo ROS. Así, la *Kinect* captura las nubes de puntos, para transferirlas al programa de visión. Este programa, tras los diferentes algoritmos de segmentación, transmite el centroide del objeto de interés al programa central.

El programa central recibe las informaciones del robot móvil *Roomba*, y con las informaciones del centroide, envía las órdenes adecuadas al brazo robotizado y al *Roomba*.

La Ilustración 22 recapitula este funcionamiento.

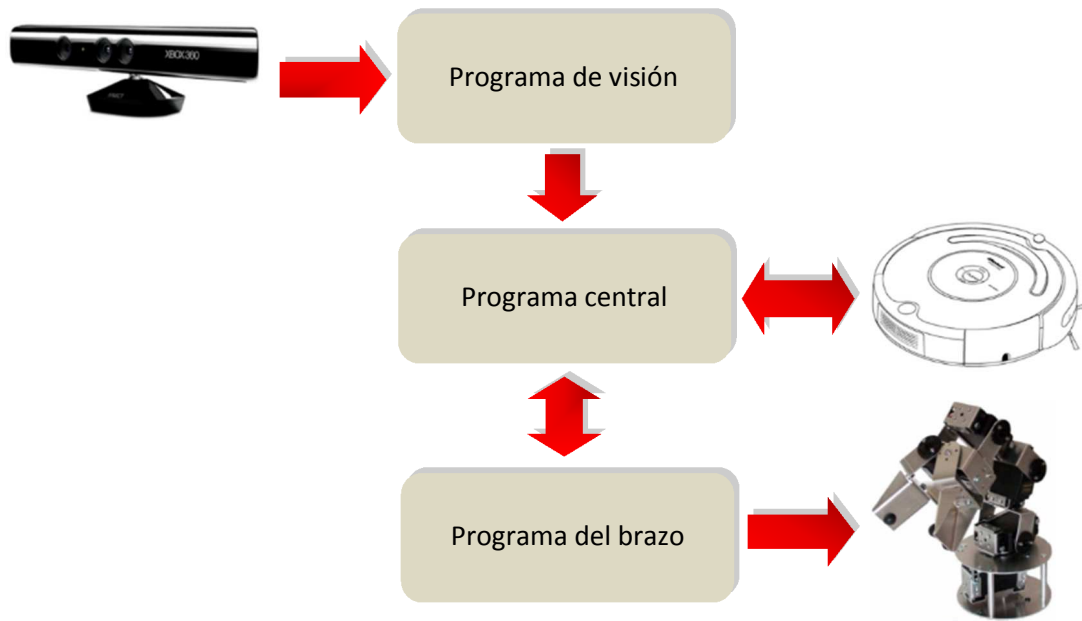
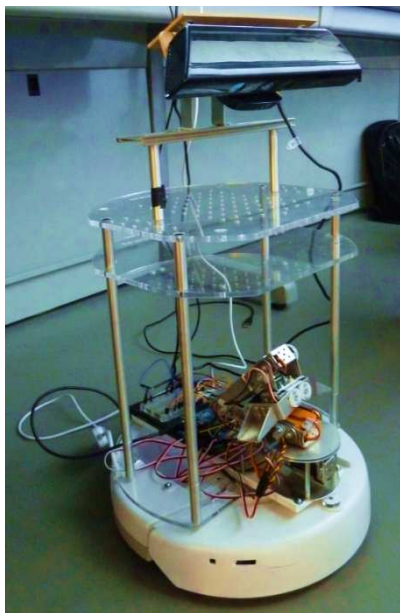


Ilustración 22: Esquema del funcionamiento del conjunto



(a)



(b)

Ilustración 23: Prototipo completo, brazo en posición inicial (a); brazo extendido (b)

4.4. Resultados

Las pruebas permiten ver que el robot ejecuta correctamente su tarea y recoge los objetos cuando están en la zona de alcanzabilidad. Puede recuperar objetos de distintos tipos (siempre con una restricción en el peso que soporta el brazo), desde algunos pequeños,

como un rollo de cinta adhesiva transparente, a alguno más grande como un pañuelo de tela.

Para obtener resultados cuantificables del sistema diseñado, como los errores, vamos a hacer dos tipos de pruebas:

- Prueba del error de precisión en la posición del brazo
- Prueba del error del sistema de visión

4.4.1. Prueba del error del brazo

En esta prueba, vamos a medir la diferencia entre la posición de consigna y la posición real medida. Para esto vamos a utilizar un código que permite imponer una consigna de posición al brazo robotizado directamente en el terminal de *Linux*. Además, dibujamos una cuadrícula en una hoja (Ilustración 24), para facilitar las medidas.



Ilustración 24: Robot en la cuadrícula, prueba del brazo

El centro de la cuadrícula coincide con el centro del sistema de coordenadas del brazo robotizado. Así todas las medidas obtenidas están calculadas en la base del brazo robotizado.

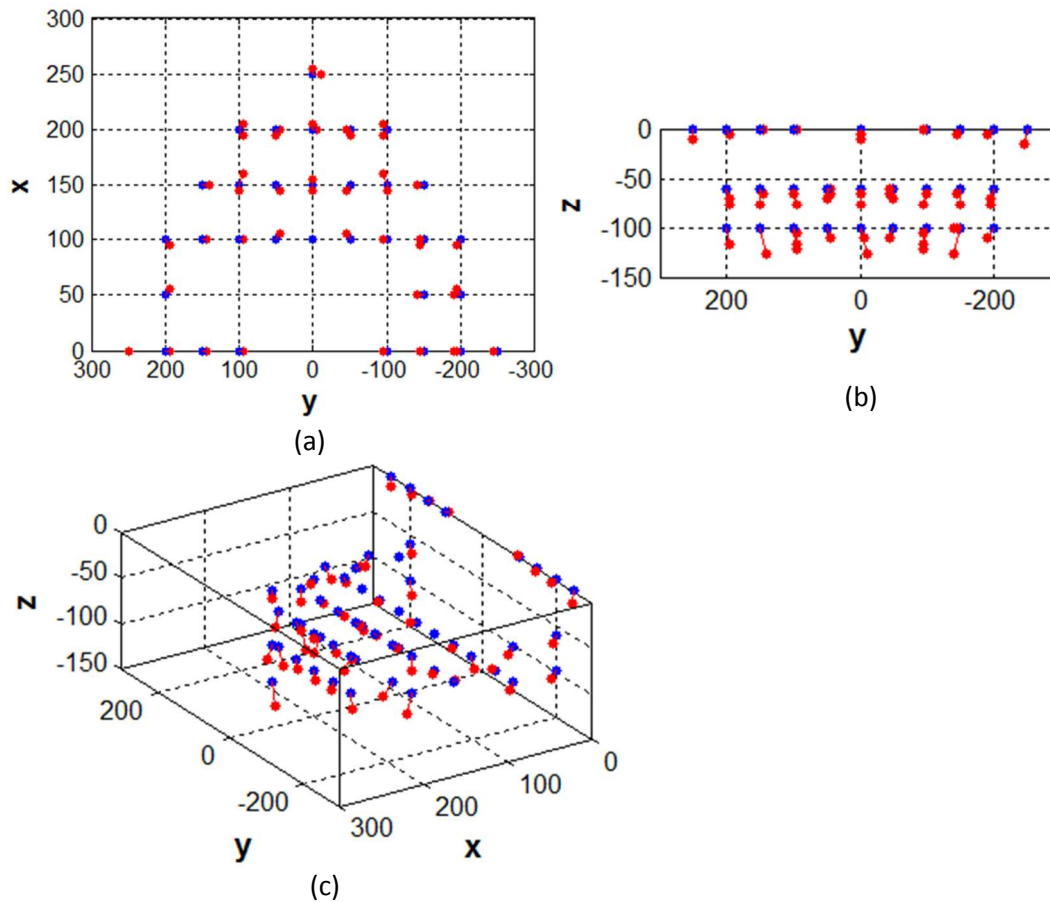


Ilustración 25: Diferencia entre la consigna (en azul) y la posición medida del brazo (en rojo). Vista de planta (a); en el plano X-Z (frente) (b); en perspectiva (c)

De estos resultados (Ilustración 25), podemos destacar dos puntos:

- El primero es que el error en Z es mucho más grande que en las otras dimensiones. Esto lo podemos imputar a la fuerza de gravedad que atrae el robot hacia el suelo generando más error en esas medidas.
- El segundo, notamos que cuanto más grande es la distancia entre el origen del robot y su posición, más grande es el error en Z, lo que confirma la hipótesis anterior (porque con la distancia aumenta el par en los servomotores).

Calculamos la media de los errores en cada eje de 57 ejecuciones y obtenemos los resultados siguientes (resultados en milímetros):

	Media del error	Error medio	Error Min	Error Max	Desviación
X	0	2,3	-10	5	3,7
Y	-0,4	3,9	-10	10	5,1
Z	8,4	8,5	0	25	7,0
d	2,8	4,1	-6	10	4,1

Tabla 1 : Parámetros del error del brazo robotizado

Vemos (Tabla 1) que estos errores son muy pequeños y que los únicos valores significativos son el error en Z que es casi del orden del centímetro y el error en d (sobre el diámetro) que es del orden del milímetro.

4.4.2. Prueba del error del sistema de visión

En esta prueba, lo primero vamos a medir la diferencia entre la posición del centroide detectada por el algoritmo de visión y su posición real medida. Para esto vamos a modificar el código de visión para que proporcione las coordenadas del centroide del objeto en un terminal de *Linux*. Utilizamos la cuadrícula anterior para obtener las medidas del centroide.

Como en la prueba anterior, el centro de la cuadrícula coincide con el centro del sistema de coordenadas del brazo robotizado. Así todas las medidas obtenidas están calculadas en la base del brazo robotizado (Ilustración 26).

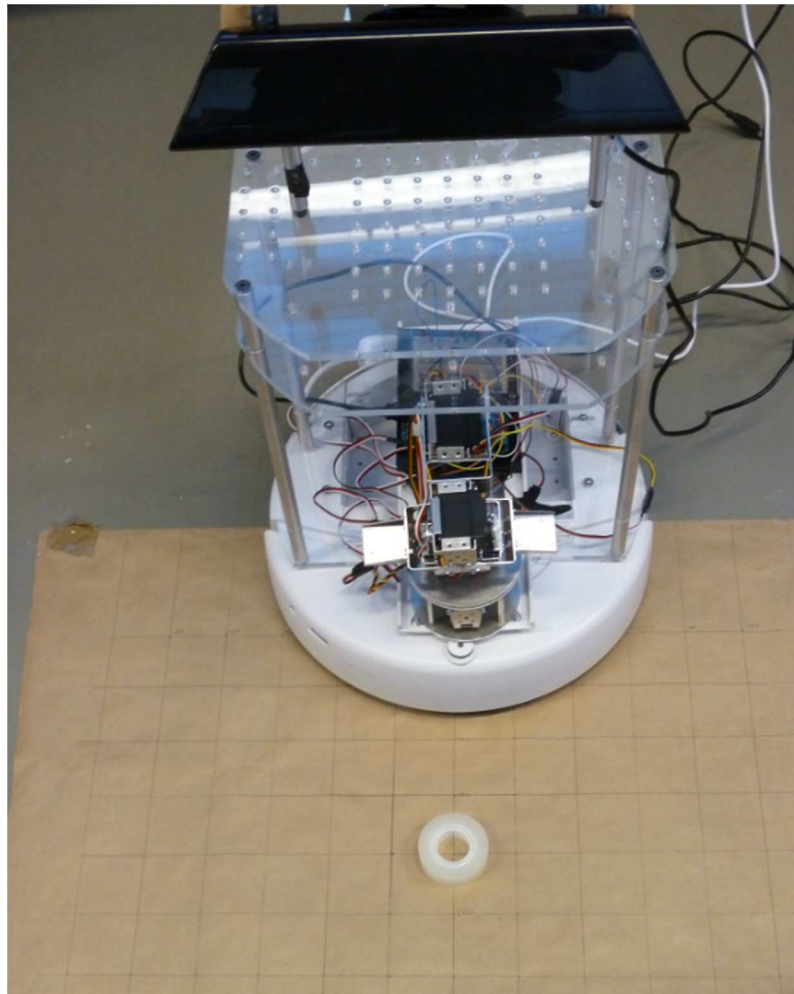


Ilustración 26: Robot en la cuadrícula, prueba de visión

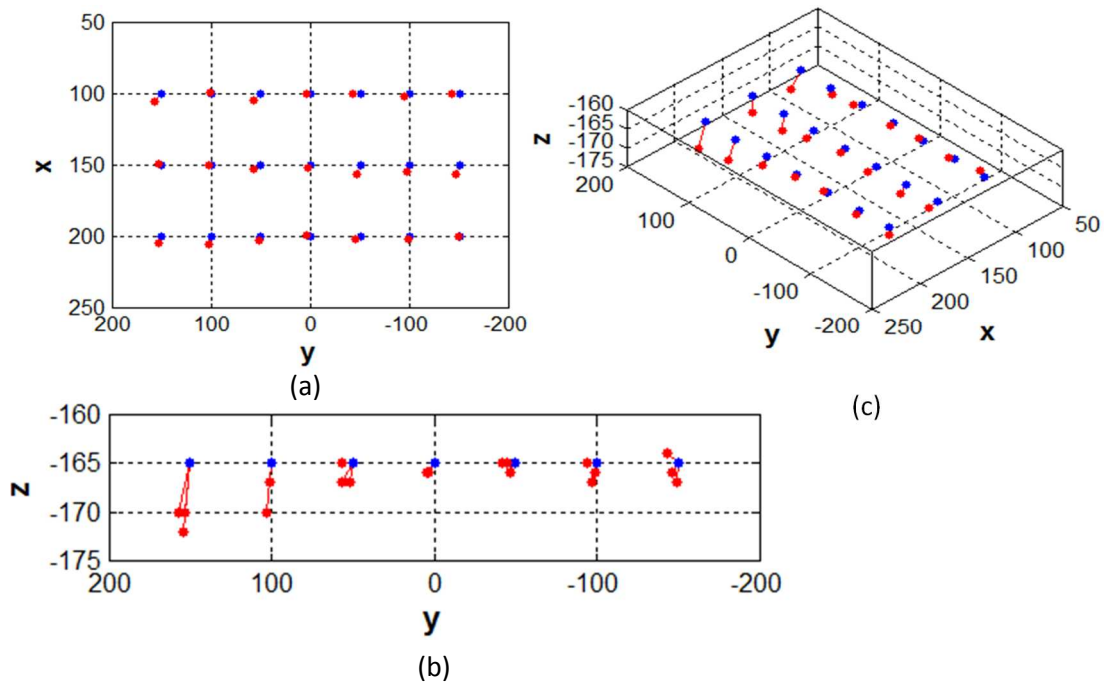


Ilustración 27: Diferencia entre la posición del centroide (en azul) y su posición calculada por el algoritmo de visión (en rojo). Vista de planta, plano X-Y (a); de frente, plano Y-Z (b); en perspectiva (c)

De estos resultados (Ilustración 27), podemos destacar dos puntos:

- El primero es que el error en Z varía en función de la posición en Y, es decir que en realidad, la cámara no está perfectamente paralela al plano del suelo
- El segundo, notamos que los errores en Y son siempre negativos y parecen constantes, lo que implica que nuestra transferencia de coordenadas entre la *Kinect* y el brazo robotizado puede ser mejorada.

Calculamos la media de los errores en cada eje de 30 ejecuciones y obtenemos los resultados siguientes (resultados en milímetros):

	Media del error	Error medio	Error Min	Error Max	Desviación
X	-2,3	2,5	-6	1	2,5
Y	-4,1	4,1	-8	-1	2,2
Z	2,1	2,0	-1	7	2,2

Tabla 2: Parámetros del error del sistema de visión

Vemos (Tabla 2) que estos errores son muy pequeños (del orden del milímetro), y podríamos reajustar el código, modificando los parámetros de la transferencia de sistema de coordenadas entre el sistema de la *Kinect* y el del brazo. Pero incluso podríamos también imputar los errores en X y Y a errores de medidas, por la poca precisión de nuestro sistema de evaluación.

4.4.3. Límites de alcance

Medimos también los parámetros de alcance de nuestro robot. Los resultados están consignados en la tabla siguiente (Tabla 3):

Elementos	Medida	Min	Max
Sistema de visión	Límite en X en B_{br} (mm)	0	1200
	Límite en Y en B_{br} (mm)	-350	350
	Límite en Z en B_{br} (mm)	-170	200
Zona de detección	Límite en X en B_{br} (mm)	10	220
	Límite en Y en B_{br} (mm)	-170	170
	Límite en Z en B_{br} (mm)	-170	0
Brazo robotizado	Distancia d (mm)	10	300
	Límite en X en B_{br} (mm)	0	300
	Límite en Y en B_{br} (mm)	-300	300
	Límite en Z en B_{br} (mm)	-170	300
Objeto	Tamaño (puntos)	90	2000
	Tamaño (cm ²)	15	350

Tabla 3: Parámetros del prototipo

Nota: B_{br} , base del brazo robotizado.

5. Conclusión

5.1. Conclusiones del trabajo y trabajo futuro

Para concluir, podemos decir que el robot responde correctamente a los objetivos iniciales del proyecto:

- El prototipo detecta e identifica los objetos que le molestan en el cumplimiento de su tarea de limpieza mediante un sensor RGB-d (sensores de visión y de profundidad) como por ejemplo el sensor *Kinect* de *Microsoft*.
- Una vez detectado el objeto, y si corresponde a un objeto que se puede recoger y que está situado en la zona de alcanzabilidad del brazo robotizado, el robot lo recoge permitiéndole continuar el cumplimiento de su tarea de limpieza.

El prototipo final no solo efectúa las tareas impuestas sino que las hace con una buena precisión, tanto desde el punto de vista del sistema de visión como del brazo robotizado, lo que le permite recuperar objetos de pequeño tamaño como un rollo de cinta adhesiva transparente.

Sin embargo, aunque el robot coge perfectamente objetos “flexibles” y manejables (como los pañuelos u objetos de tela), tiene más dificultades con los objetos rígidos. Esto es debido a dos cosas: la primera es que el robot solo se fija en un punto para recoger los objetos, dirigiéndose hacia su centroide. Entonces, si el objeto rígido no está posicionado adecuadamente, la pinza no consigue agarrar el objeto, lo que implica que no puede moverlo. Otra limitación está en el propio funcionamiento de la garra, que se cierra de manera angular y no paralelamente.

Por consecuencia, la parte que podemos mejorar es este aspecto del funcionamiento del robot añadiendo módulos de *grasping* más avanzados, que permiten determinar la mejor manera de recoger el objeto y los puntos de agarre óptimos a los que dirigir la pinza. Para poder incluir un módulo *grasping* más avanzado tenemos que modificar también la muñeca del brazo robotizado para dotarla de grados de libertad suplementarios.

Otras mejoras se podrían hacer en trabajos futuros, como la implementación o integración de un módulo de reconocimiento de objetos en el algoritmo de visión, que permitiría ordenar los objetos recogidos, o el cambio de los actuadores para que sean más potentes y que puedan devolver las informaciones de posición.

De aspecto general, vemos que el robot no está dimensionado para una aplicación concreta. Su altura lo hace demasiado voluminoso para que pueda ser utilizado como aspiradora en una casa y la estructura del brazo es demasiado débil. Además, el brazo tiene poco alcance por sus dimensiones y su posición actuales.

No obstante, si modificamos estos elementos tenemos que modificar los componentes del robot en su conjunto. Si deseamos reducir la altura del robot, tenemos que pensar que este

parámetro depende de la Kinect y del brazo robotizado. De la Kinect, porque esta cámara RGB-d tiene un campo mínimo de visión de 1,2 metros. Deberíamos entonces reemplazar este componente por otro que requiera un campo mínimo inferior (como los utilizados en otras cámaras de tipo RGB-d).

5.2. Conclusión personal

Por un lado, este proyecto me ha permitido tratar con las problemáticas de la robótica actual como los sistemas de visión y el control de miembros robotizados, mejorando así mis conocimientos en este campo. La libertad de maniobra y de decisión acordada por mis tutores no solo me permitió orientar el proyecto como me gustaba más, sino también implicarme totalmente en él y adquirir una cierta autonomía en mi trabajo. La experiencia ganada me servirá para mi futuro laboral en este dominio.

Por otro lado, una observación sobre el funcionamiento de la escuela, es que me parece demasiado rígida la organización de los talleres, y que no sean agrupados en un “super-taller”, que ofrecería más oportunidades en lo que se refiere a la fabricación y al uso de las máquinas de fabricación. Lo veo como una oportunidad a explorar porque la situación actual limita las posibilidades de creación y de desarrollo en los proyectos de los estudiantes.

Bibliografía

- [1] J. Jones, «Robots at the tipping point: the road to irobot Roomba,» *IEEE Robotics & Automation Magazine*, nº 13, p. 76–78, 2006.
- [2] P. Teikari, R. P. Najjar, H. Malkki, K. Knoblauch, D. Dumortier, C. Gronfiera y H. M. Coopera, «An inexpensive Arduino-based LED stimulator system for vision research,» *Journal of Neuroscience Methods*, p. 227–236, 15 November 2012.
- [3] J. A. Kornuta, M. E. Nipper y J. B. Dixon, «Low-cost microcontroller platform for studying lymphatic biomechanics in vitro,» *Journal of Biomechanics*, vol. 46, nº 1, p. 183–186, 2012.
- [4] P. Henry, M. Krainin, E. Herbst, X. Ren y D. Fox, «RGB-D mapping: Using Kinect-style depth cameras for dense 3D modeling of indoor environments,» *INTERNATIONAL JOURNAL OF ROBOTICS RESEARCH*, pp. 305-313, April 2012.
- [5] B. T. M. Leong, S. M. Low y M. P.-L. Ooi, «Low-Cost Microcontroller-based Hover Control Design of a Quadcopter,» *Procedia Engineering*, vol. 41, p. 458–464, 2012.
- [6] M. A. K. Yusoff, R. E. Samin y B. S. K. Ibrahim, «Wireless Mobile Robotic Arm,» *Procedia Engineering*, vol. 41, p. 1072–1078, 2012.
- [7] J.-J. Hernández-López, A.-L. Quintanilla-Olvera, J.-L. López-Ramírez, F.-J. Rangel-Butanda, M.-A. Ibarra-Manzano y D.-L. Almanza-Ojeda, «Detecting objects using color and depth segmentation with Kinect sensor,» *Procedia Technology*, vol. 3, p. 196–204, 2012.
- [8] M. Quigley, B. Gerkey, K. Conley, J. Faust, T. Foote, J. Leibs, E. Berger y R. Wheeler, «ROS: an open source robot operating system,» de *Proceedings of the 2009 International Conference on Robotics and Automation, Open-Source Software Workshop*, 2009.
- [9] R. Guido, S. Fitzgerald y D. Cuartielles, «Arduino Uno,» Arduino, November 2011. [En línea]. Available: <http://arduino.cc/en/Main/arduinoBoardUno>.
- [10] Y. NAKAMURA y H. HANAFUSA, «Inverse kinematic solutions with singularity robustness for robot manipulator control,» *Journal of dynamic systems, measurement, and control*, vol. 108, nº 3, pp. 163-171, 1986.
- [11] R.B Rusu, Willow Garage, «3D is here: Point Cloud Library (PCL),» de *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, 2011.

- [12] EasyRobotic, «EasyRobotic,» A. Viklund, 2006. [En línea]. Available: <http://www.easyrobotics.fr/>. [Último acceso: 20 01 2013].
- [13] R. Firoozian, *Servo Motors and Industrial Control Theory*, 1 ed., Springer, 2008.
- [14] A. KASINSKI, «TRAJECTORY PLANNING AND COMPUTER-ANALYSIS OF KINEMATIC PROBLEMS FOR MANIPULATORS,» *SYSTEMS ANALYSIS MODELLING SIMULATION*, vol. 8, nº 9, pp. 715-719, 1991.
- [15] A. F. Martin y C. B. Robert, «Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography,» *Communications of the ACM*, vol. 24, nº 6, pp. 381 - 395, 1981.
- [16] J. A. Hartigan, *Clustering Algorithms*, New York: John Wiley & Sons, 1975.
- [17] J. L. Bentley, «Multidimensional binary search trees used for associative searching,» *Communications of the ACM*, vol. 18, nº 9, pp. 509 - 517 , 1975.
- [18] W. Garage, «Willow Garage,» 2011. [En línea]. Available: <http://www.willowgarage.com/turtlebot>. [Último acceso: November 2012].

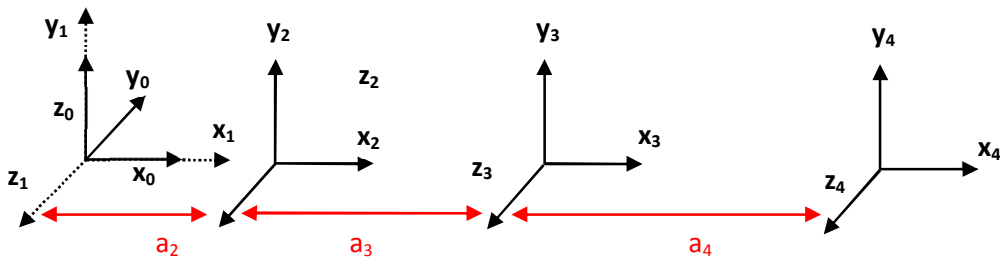
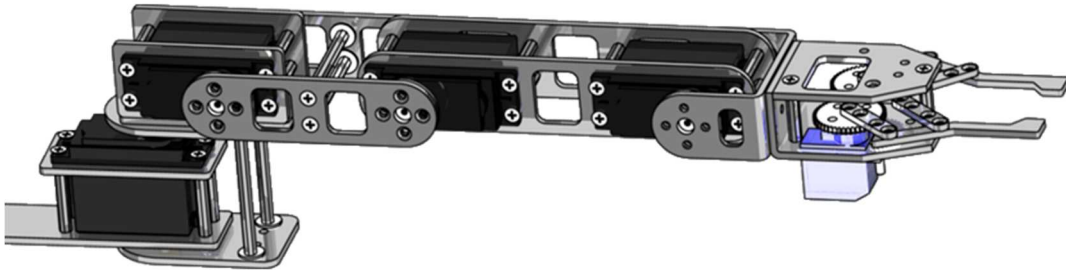
Tabla de las ilustraciones

Ilustración 1: Diagrama temporal del trabajo.....	6
Ilustración 2: Renderizado de la vista 3D de CatiaV5.....	8
Ilustración 3: Descomposición de <i>brick</i> EasyRobotic.....	9
Ilustración 4: Ejemplo de robot construido con kits EasyRobotic.....	9
Ilustración 5: Brazo robotizado construido con la alternativa elegida.....	10
Ilustración 6: Servomotor Hitec HS-311.....	11
Ilustración 7: Vista del interior del servo-motor.....	11
Ilustración 8: Plaqueta de microcontrolador Arduino Uno.....	13
Ilustración 9 : Descripción del calibrador.....	14
Ilustración 10: Curva de conversión de grados a impulsión.....	15
Ilustración 11 : Descripción de la Kinect.....	17
Ilustración 12: imagen RGB (a) e imagen de infrarrojos (b).....	18
Ilustración 13: Esquema del funcionamiento del detector de profundidad.....	18
Ilustración 14 : imagen RGB (a) e imagen de profundidad (b).....	19
Ilustración 15: Visualizador Rviz.....	20
Ilustración 16: (a) Nube de puntos y (b) nube de puntos después del <i>voxel-grid</i>	21
Ilustración 17: (a) Superposición de la detección con la imagen RGB, (b) reconstrucción en 3D. En azul, el plano del suelo detectado; en rosa, los otros puntos captados sobrantes. ..	21
Ilustración 18: (a) Superposición de la detección de objetos con la imagen RGB, (b) reconstrucción en 3D. En verde, los objetos detectados por la agrupación. En azul, el plano del suelo detectado. En rosa, los otros puntos captados sobrantes.....	22
Ilustración 19: (a) Superposición de la detección con la imagen RGB, (b) reconstrucción en 3D. En amarillo, el objeto de interés. En verde, los objetos detectados por la agrupación. En azul, el plano del suelo detectado. En rosa, los otros puntos captados sobrantes.	23
Ilustración 20: Ejes de la Kinect.....	23
Ilustración 21: Esquema de la transferencia de coordenadas, R_k sistema de coordenadas de la Kinect, y R_{br} de coordenadas del brazo robotizado.	24
Ilustración 22: Esquema del funcionamiento del conjunto.....	26
Ilustración 23: Prototipo completo, brazo en posición inicial (a); brazo extendido (b).....	26
Ilustración 24: Robot en la cuadrícula, prueba del brazo.....	27
Ilustración 25: Diferencia entre la consigna (en azul) y la posición medida del brazo (en rojo). Vista de planta (a); en el plano X-Z (frente) (b); en perspectiva (c).....	28
Ilustración 26: Robot en la cuadrícula, prueba de visión.....	29
Ilustración 27: Diferencia entre la posición del centroide (en azul) y su posición calculada por el algoritmo de visión (en rojo). Vista de planta, plano X-Y (a); de frente, plano Y-Z (b); en perspectiva (c).....	30

Anexos

Anexo I: Calculo del modelo inverso	I
Anexo II: El antiguo brazo robotizado.....	V
Anexo III: Presupuestos	VI
Anexo IV: Planos de las piezas.....	XI
Anexo V: Renderizados del brazo robotizado	XXIII
Anexo VI: Resultados de la calibración	XXVI
Anexo VII: Especificaciones técnicas de la Kinect.....	XXIX
Anexo VIII: Fabricación de piezas “artesanales”	XXX
Anexo IX: Manual de Usuario del sistema	XXXIII

Anexo I: Calculo del modelo inverso



Nb: La imagen no corresponde al robot que hemos usado en nuestro caso. Esta imagen permite ilustrar los parámetros del DH. Sin embargo la posición inicial es la misma.

Conociendo los parámetros del modelo geométrico de nuestro robot, podemos deducir los parámetros DH (Denavit–Hartenberg) siguientes:

	a_i (mm)	α_i (rad)	d_i (mm)	Θ_i (rad)
1	0	$\pi/2$	0	0
2	110	0	0	0
3	110	0	0	0
4	105	0	0	0

Lo que nos da la matriz de DH de matlab:

$$nono = \begin{bmatrix} \frac{\pi}{2} & 0 & 0 & 0 & 0 \\ 0 & 110 & 0 & 0 & 0 \\ 0 & 110 & 0 & 0 & 0 \\ 0 & 105 & 0 & 0 & 0 \end{bmatrix}$$

nono=($\alpha_i, a_i, \Theta_i, d_i, \sigma_i$) con $\sigma_i=0$ por un eje de rotación y 1 por un eje de translación

Buscamos el modelo geométrico inverso (Inverso Kinematic o ikine) de nuestro robot. Para eso, utilizamos las matrices U y V.

En nuestro caso, la matriz U1 bale:

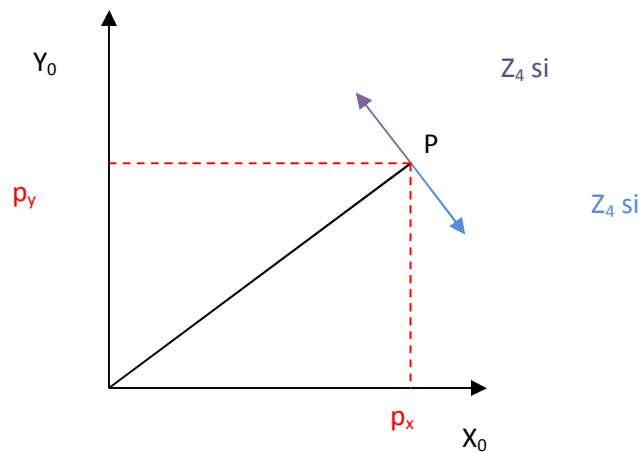
$$U1 = \begin{bmatrix} \cos(t2 + t3 + t4) * \cos(t1) & -\sin(t2 + t3 + t4) * \cos(t1) & \sin(t1) & U214 * \cos(t1) \\ \cos(t2 + t3 + t4) * \sin(t1) & -\sin(t2 + t3 + t4) * \sin(t1) & -\cos(t1) & U214 * \sin(t1) \\ \sin(t2 + t3 + t4) & \cos(t2 + t3 + t4) & 0 & U224 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\text{Con } U214 = a3 * \cos(t2 + t3) + a2 * \cos(t2) + a4 * \cos(t2 + t3 + t4)$$

$$\text{y } U224 = a3 * \sin(t2 + t3) + a2 * \sin(t2) + a4 * \sin(t2 + t3 + t4)$$

$$U1 = V0 = \begin{pmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{pmatrix}$$

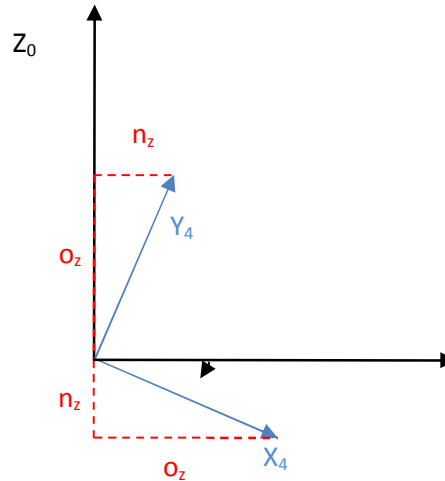
Así, tenemos $t1 = \text{atan2}(a_x, -a_y)$ o $t1 = \text{atan2}\left(\frac{p_y}{U214}, \frac{p_x}{U214}\right) = \text{atan2}(p_y, p_x)$ o $t1 = \text{atan2}(-p_y, -p_x)$



Vemos también que, ahora que conocemos t1, tenemos un sistema resoluble:

$$\begin{cases} \sin(t2 + t3 + t4) = n_z \\ \cos(t2 + t3 + t4) = o_z \end{cases}$$

$$t2 + t3 + t4 = \text{atan2}(n_z, o_z)$$



$$\begin{cases} (a3 * \cos(t2 + t3) + a2 * \cos(t2) + a4 * \cos(t2 + t3 + t4)) * \cos(t1) = p_x \\ a3 * \sin(t2 + t3) + a2 * \sin(t2) + a4 * \sin(t2 + t3 + t4) = p_z \end{cases}$$

$$\begin{cases} a3 * \cos(t2 + t3) + a2 * \cos(t2) = \frac{p_x}{\cos(t1)} - a4 * o_z = p_1 \\ a3 * \sin(t2 + t3) + a2 * \sin(t2) = p_z - a4 * n_z = p_2 \end{cases}$$

$$\begin{cases} (a3 * \cos(t2 + t3))^2 = (p_1 - a2 * \cos(t2))^2 \\ (a3 * \sin(t2 + t3))^2 = (p_2 - a2 * \sin(t2))^2 \end{cases}$$

$$\begin{cases} a3^2 * \cos^2(t2 + t3) = p_1^2 + a2^2 * \cos^2(t2) - 2 * a2 * p_1 * \cos(t2) \\ a3^2 * \sin^2(t2 + t3) = p_2^2 + a2^2 * \sin^2(t2) - 2 * a2 * p_2 * \sin(t2) \end{cases}$$

$$p_1 * \cos(t2) + p_2 * \sin(t2) = \frac{p_1^2 + p_2^2 + a2^2 - a3^2}{2 * a2} = c$$

Lo que nos da: $t2 = \text{atan2}(p_2, p_1) \pm \arccos\left(\frac{c}{\sqrt{p_1^2 + p_2^2}}\right)$

Ahora que tenemos t2, con el sistema de ecuaciones precedente podemos determinar t2+t3

$$\begin{cases} a3 * \cos(t2 + t3) = p_1 - a2 * \cos(t2) \\ a3 * \sin(t2 + t3) = p_2 - a2 * \sin(t2) \end{cases}$$

Lo que nos da: $t3 = \text{atan2}(p_2 - a2 * \sin(t2), p_1 - a2 * \cos(t2)) - t2$

$$t2 + t3 + t4 = \text{atan2}(n_z, o_z)$$

$$t4 = \text{atan2}(n_z, o_z) - t3 - t2$$

Resultado final:

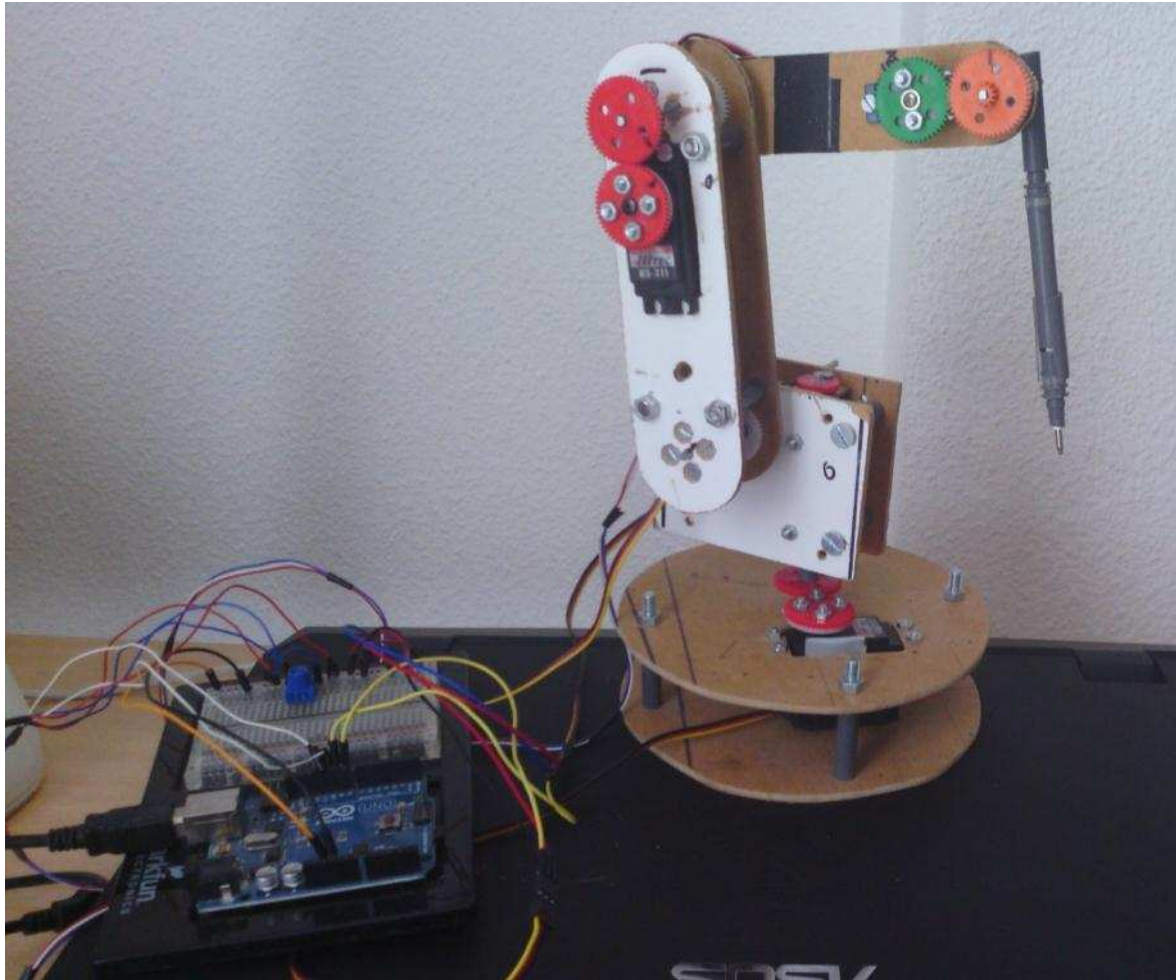
$$t1 = \text{atan2}(p_y, p_x) \text{ o } t1 = \text{atan2}(-p_y, -p_x)$$

$$t2 = \text{atan2}\left(p_z - a4 * n_z, \frac{p_x}{\cos(t1)} - a4 * o_z\right) \pm \arccos\left(\frac{\left(\frac{p_x}{\cos(t1)} - a4 * o_z\right)^2 + (p_z - a4 * n_z)^2 + a2^2 - a3^2}{2 * a2 * \sqrt{\left(\frac{p_x}{\cos(t1)} - a4 * o_z\right)^2 + (p_z - a4 * n_z)^2}}\right)$$

$$t3 = \text{atan2}\left((p_z - a4 * n_z - a2 * \sin(t2), \frac{p_x}{\cos(t1)} - a4 * o_z\right) - t2$$

$$t4 = \text{atan2}(n_z, o_z) - t3 - t2$$

Anexo II: El antiguo brazo robotizado















El antiguo brazo robotizado está hecho de chapa de madera reciclada. Su herramienta era un bolígrafo que permitía dibujar (con poca precisión) en una hoja de papel en el suelo.

Integraba engranajes para transmitir los movimientos y limitar los movimientos a la rotación sobre el eje. El problema era que los pares en las conexiones entre los ejes y los engranajes eran demasiado importantes lo que producía de vez en cuando deslizamientos. Estos deslizamientos cambiaban las posiciones de los miembros del brazo, lo que implicaba errores de posición.

Otro problema: la madera utilizada era demasiado flexible y pesaba demasiado, lo que aumentaba los errores de posición.

Anexo III: Presupuestos



Article	Quantity	Price	Sum
 10-pack of M3 25x7 spacer bolts [Product ID: 521676-I5] Delivery status: In stock	2 pack	2,68 €	5,36 €
 ALUMINIUM-PLATES AL99,5 400x200x1.5 MM [Product ID: 293091-I5] Delivery status: In stock	2 unit(s)	7,65 €	15,30 €
 10-pack of M3 50x7 spacer bolts [Product ID: 521740-I5] Delivery status: In stock	1 pack	4,09 €	4,09 €
 100-pck DIN 965 M3/10 countersunk screws [Product ID: 522235-I5] Delivery status: In stock	1 pack	3,21 €	3,21 €
 Coll. bush 3 [Product ID: 237027-I5] Delivery status: In stock	5 unit(s)	1,15 €	5,75 €
 TOOLCRAFTDIN 934 hex nuts A2 M310 Units [Product ID: 888118-I5] Delivery status: In stock	2 pack	0,95 €	1,90 €
 100-pack of DIN9021 steel M3 shims [Product ID: 521800-I5] Delivery status: In stock	1 pack	1,23 €	1,23 €
 Hitec Standard servo HS-311Storage -Gears Polyamide JR [Product ID: 209893-I5] Delivery status: In stock	1 unit(s)	7,95 €	7,95 €
 Set collars 3mm [Product ID: 225401-I5] Delivery status: In stock	1 pack	3,95 €	3,95 €
 Double gear assortment M0.5 20 pce [Product ID: 297704-I5] Delivery status: In stock	1 set	6,95 €	6,95 €
 Quality Spring Steel Wire 3.0 Mm X1000 [Product ID: 238110-I5] Delivery status: In stock	1 unit(s)	2,75 €	2,75 €
 10-pack of M3 20x20 spacer bolts [Product ID: 521663-I5] Delivery status: In stock	1 pack	2,47 €	2,47 €
Sub total			60,91 €
Total			60,91 €
Incl. VAT			9,73 €




LASER EBRO, S.L.
zaragoza@laserebro.com www.laserebro.com

Pol. Ind. La Puebla de Alfindón,
C/ Del Chopo, parcela 88-100
50171 La Puebla de Alfindón (Zaragoza)
tel: 976 108 900 fax: 976 108 901

Atn:
PIERRE HERMAN
CIF: _____
TFNO: _____
FAX: _____

OFERTA N°: 148230 (27/09/2012)

REF:

En caso de aceptar esta oferta le rogamos que indique el numero de esta en su pedido
Valides de la oferta : 1 semana
Precios no válidos para pedidos ni entregas parciales. En caso de referencias
por separado o lotes inferiores a los indicados deberán pedir revisión de precios.

- * Forma de pago: al contado (al recoger las piezas)
- * Portes : sus medios
- * Plazo de entrega : 2 semanas

CANTIDAD	DENOMINACION	MATERIAL	Euros/ud
2	EPAULE	A2,5	1,50
1	EPAULEBIS PL	A2,5	11,80
1	AVANTBRAS	A2,5	3,21
1	AVANTBRASBIS	A2,5	6,01
2	BRAS	A2,5	6,10
2	BRASCOURT	A2,5	4,06
1	EPAULEFIXATION	A2,5	1,60
1	TETE PL	A2,5	10,89
1	CORPSPINCE PL	A2,5	10,92
4	BIELE	A2,5	3,47
1	CORPSPINCEBIS	A2,5	2,52
2	PINCE	A2,5	0,82

Total: 85,79 €

A. en su nombre:

ROBERTO MAS LEBRE



Pol. Ind. Villanueva de Gállego
Ctra. Huesca Km. 12,100 - Nave 6
50830 Villanueva de Gállego- Zaragoza (España)
Tel.: 976 185 022 Fax.: 976 180 136 C.I.F. B99050916

Presupuesto nº: 12-822

Fecha: 08-OCT-2012

CLIENTE: UNIVERSIDAD

Teléfono: _____ ATENCIÓN: RAFA LANA

Fax.: _____ e-mail: mecprec@unizar.es

Les adjunto presupuesto desglosado solicitado:

Descripción	Ud.	Precio unidad	Neto
Corte y material según fichero PDF (Plans0210-A) y DXF (Placas brazo) en Aluminio de 3mm de espesor. Corte por agua. Calidad normal.	19		76.-
TOTAL PRESUPUESTO (IVA no incluido)			76.-

Condiciones Generales: Cortenfrío S.L. no se hace responsable del material aportado por el cliente en caso de defectos en el corte, pero se repetirá el corte sin cargo.

CONDICIONES DEL PRESUPUESTO.

Material: Incluido
 Transporte: No incluido
 Validez: 15 días
 Plazo de entrega: 2-3 días laborables.
 Forma de pago: Habitual.

OBSERVACIONES:

De ser conforme el presupuesto, necesitamos nos remitan firmado y sellado vía fax.

CONFORMIDAD DE PEDIDO.

Su nº de Pedido: _____
 Fecha aceptación: _____
 Albarán de entrega valorado: Sí No
 Firma y sello:



EASY ROBOTICS

La robotique à la portée de tous

Luis Montano -
Universidad de Zaragoza
(CIF Q5018001G)
C/ Maria de Luna 1
Edif Ada Byron, DIIS
50018 Zaragoza.

Devis en Euros

Numéro	Date	Vos REF	Mode de règlement	Affaire suivit par :
DV25101201	25/10/12			Mr HERMAN























Total HT :	134,95 EUR
TVA :	26,45 EUR
Total TTC :	161,40 EUR
Frais de port : (Colissimo avec assurance)	27, 80 EUR
Total a regler :	189,20 EUR



EASY ROBOTICS

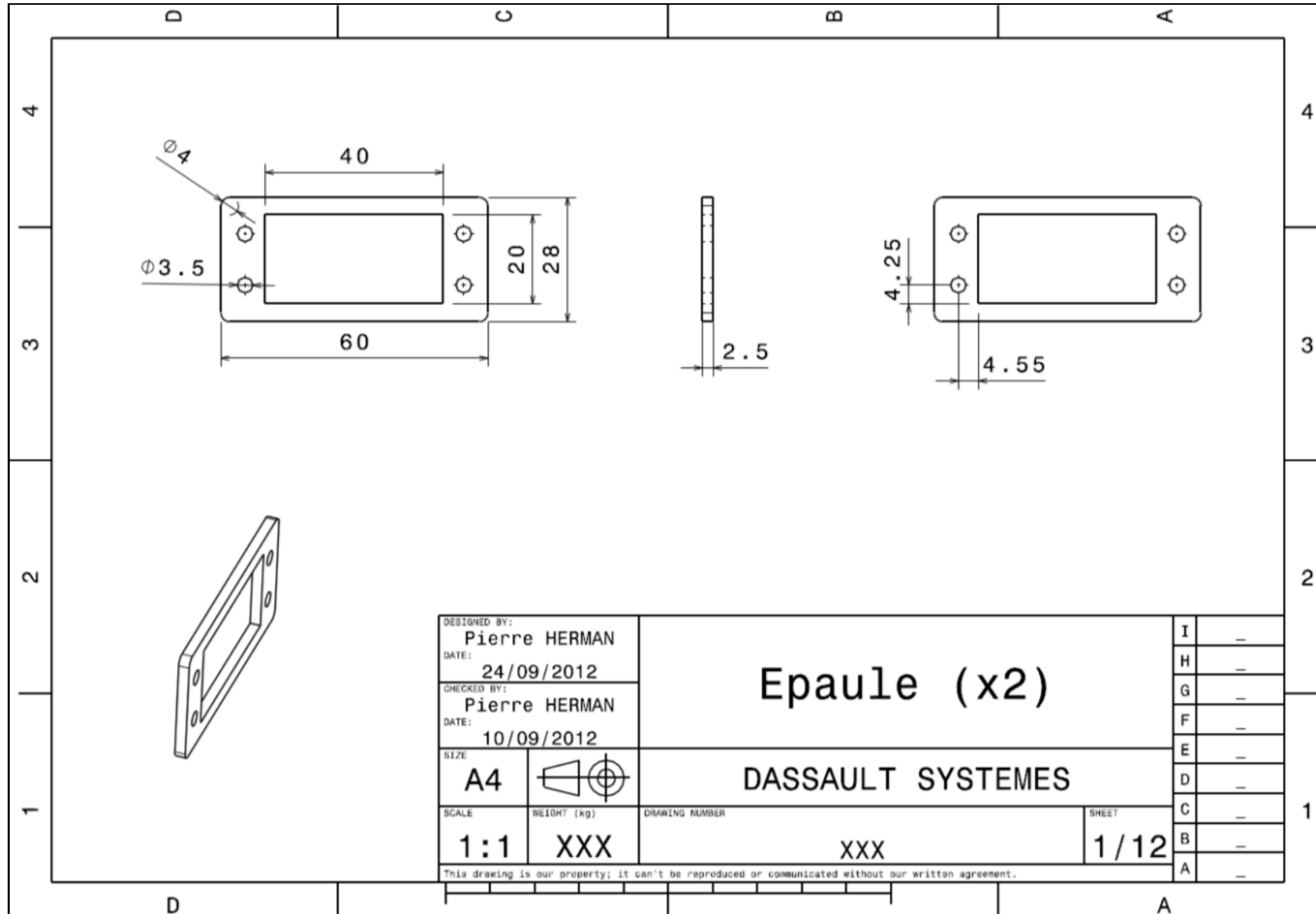
La robotique à la portée de tous

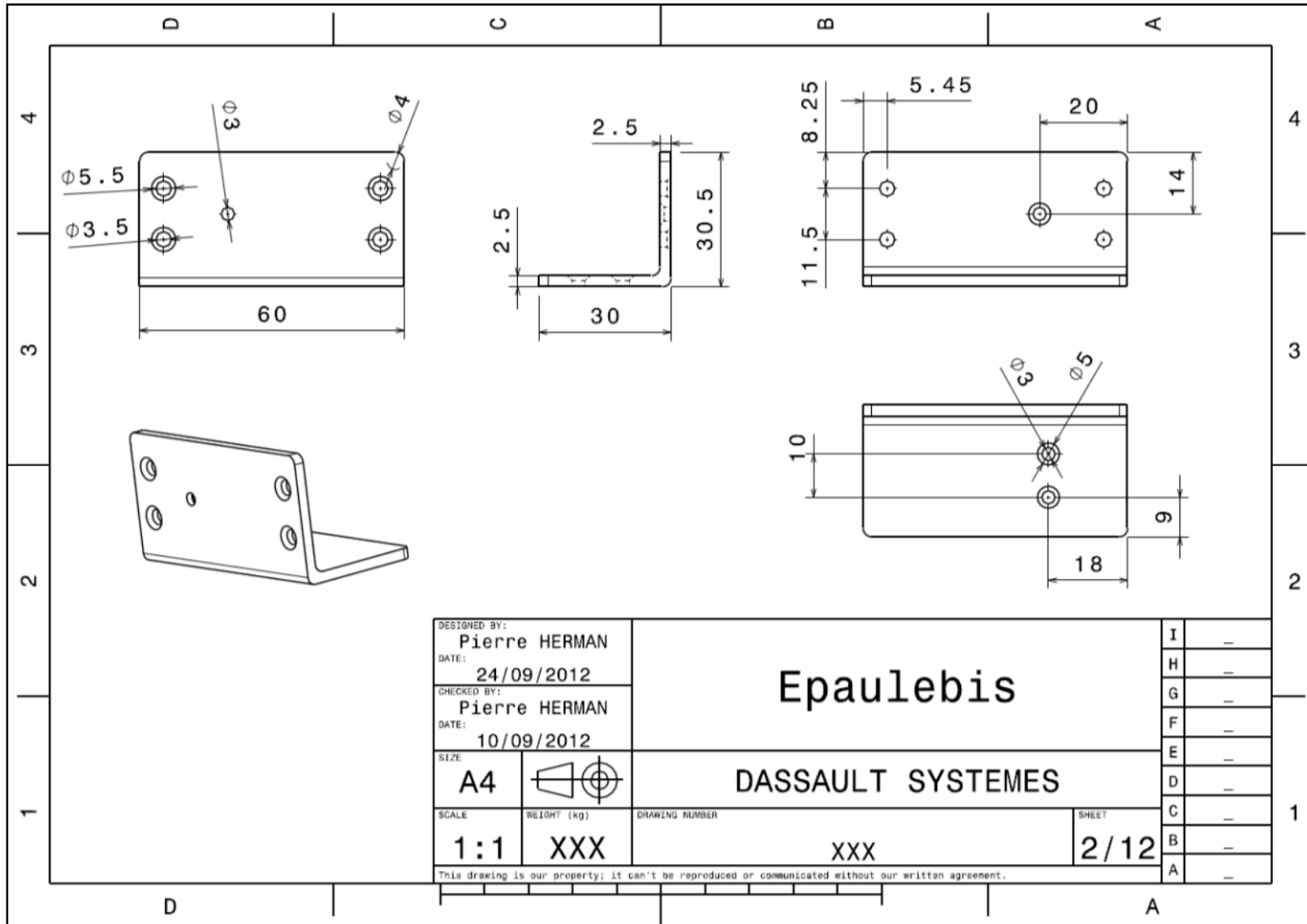
Numéro DV25101201	Date 25/10/12	Vos REF	Mode de règlement	Affaire suivit par : Mr Lesfritz
-----------------------------	------------------	---------	-------------------	-------------------------------------

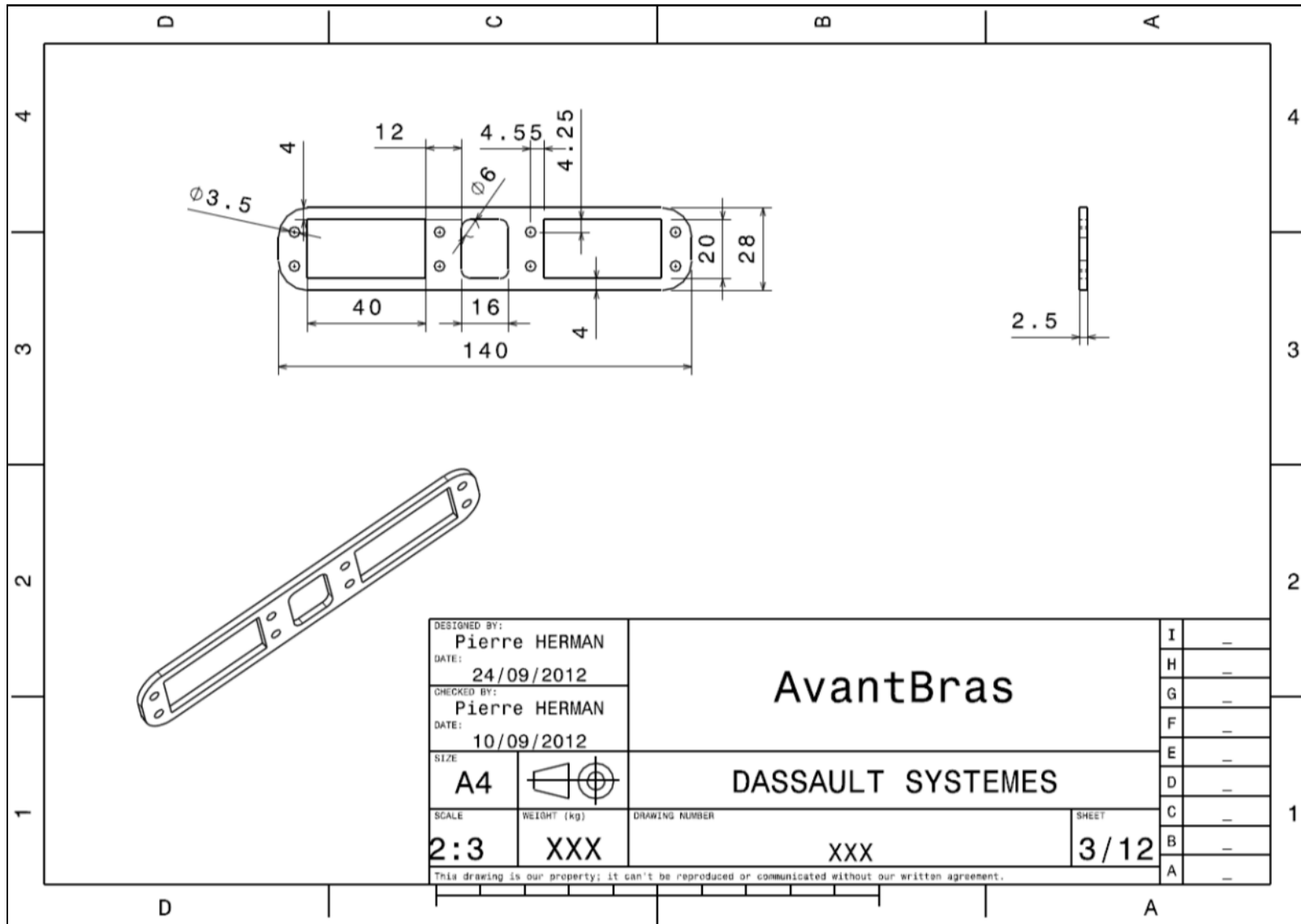
Produit	Description	Réf.	Dispo.	Prix unitaire	Qté	Total
	Brique Easy (module mecatronique)	ERBRICK	●	10,45 € HT	5 	62,50 €
	Servomoteur FUTABA S3010 Reprise : Standart	S3010	●	18,39 € HT	1 	22,00 €
	Support Brique et carte	ERSUPBR1	●	17,56 € HT	1 	21,00 €
	Kit mécanisme de pince	ERKPINC1-X	●	12,21 € HT	1 	14,60 €
	Mors de pinces	ERMPINC1-1	●	6,61 € HT	1 	7,90 €
	Rallonges Servo Longueur : 90 Cm	Rallong90	●	1,59 € HT	2 	3,80 €
	Rallonges Servo Longueur : 50 Cm	Rallong50	●	1,34 € HT	3 	4,80 €
	Fourche (Longue)	ERFOUR1-2	●	1,84 € HT	3 	6,60 €
	Fourche (Pour Bras)	ERFOUR1-3	●	2,09 € HT	1 	2,50 €
	Fourche (Classique)	ERFOUR1-1	●	1,51 € HT	4 	7,20 €
	Servomoteur FUTABA S3003 Reprise : Standart	S3003	●	7,11 € HT	1 	8,50 €
Total produits TTC :						161,40 €
Total TTC :						161,40 €

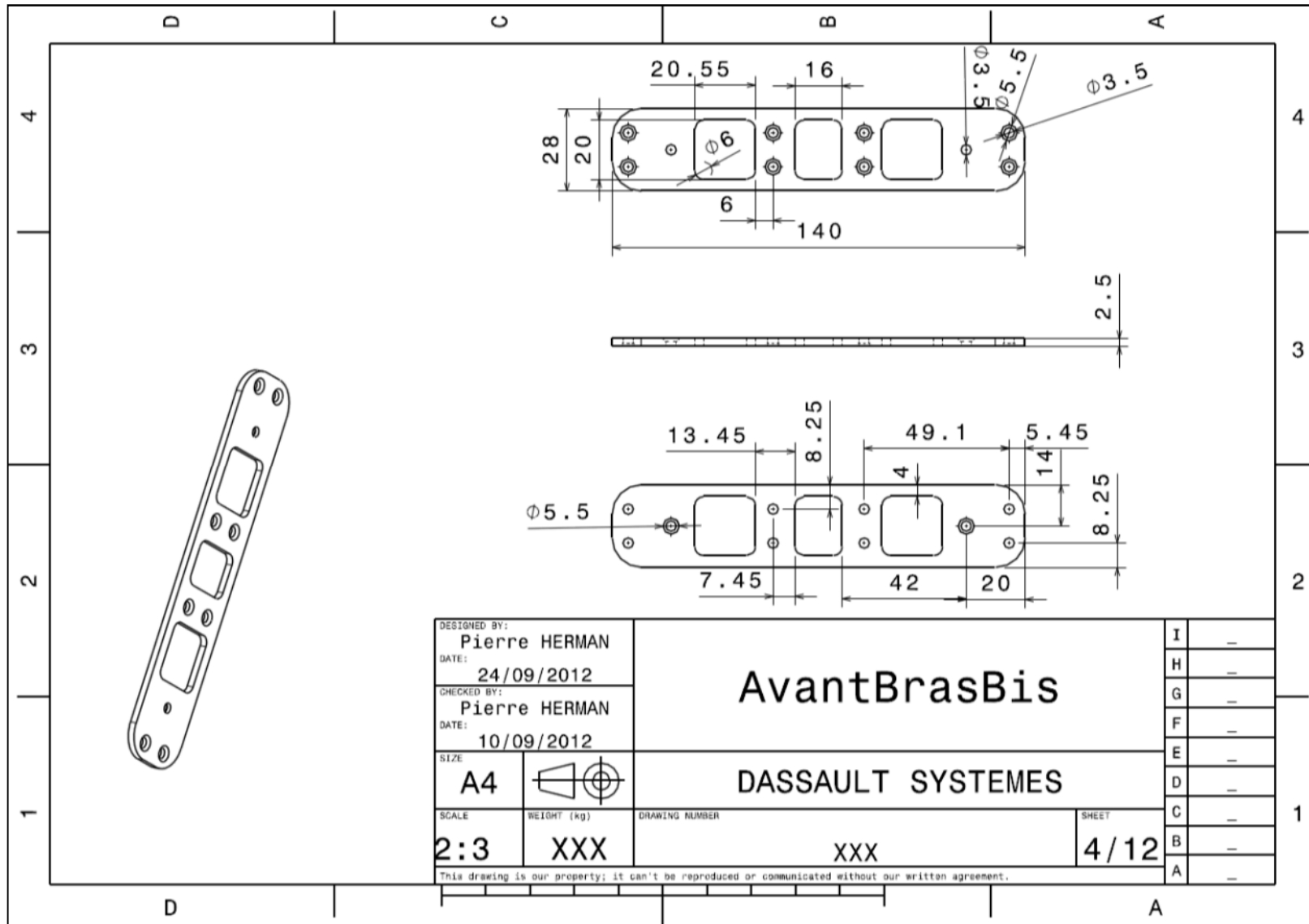


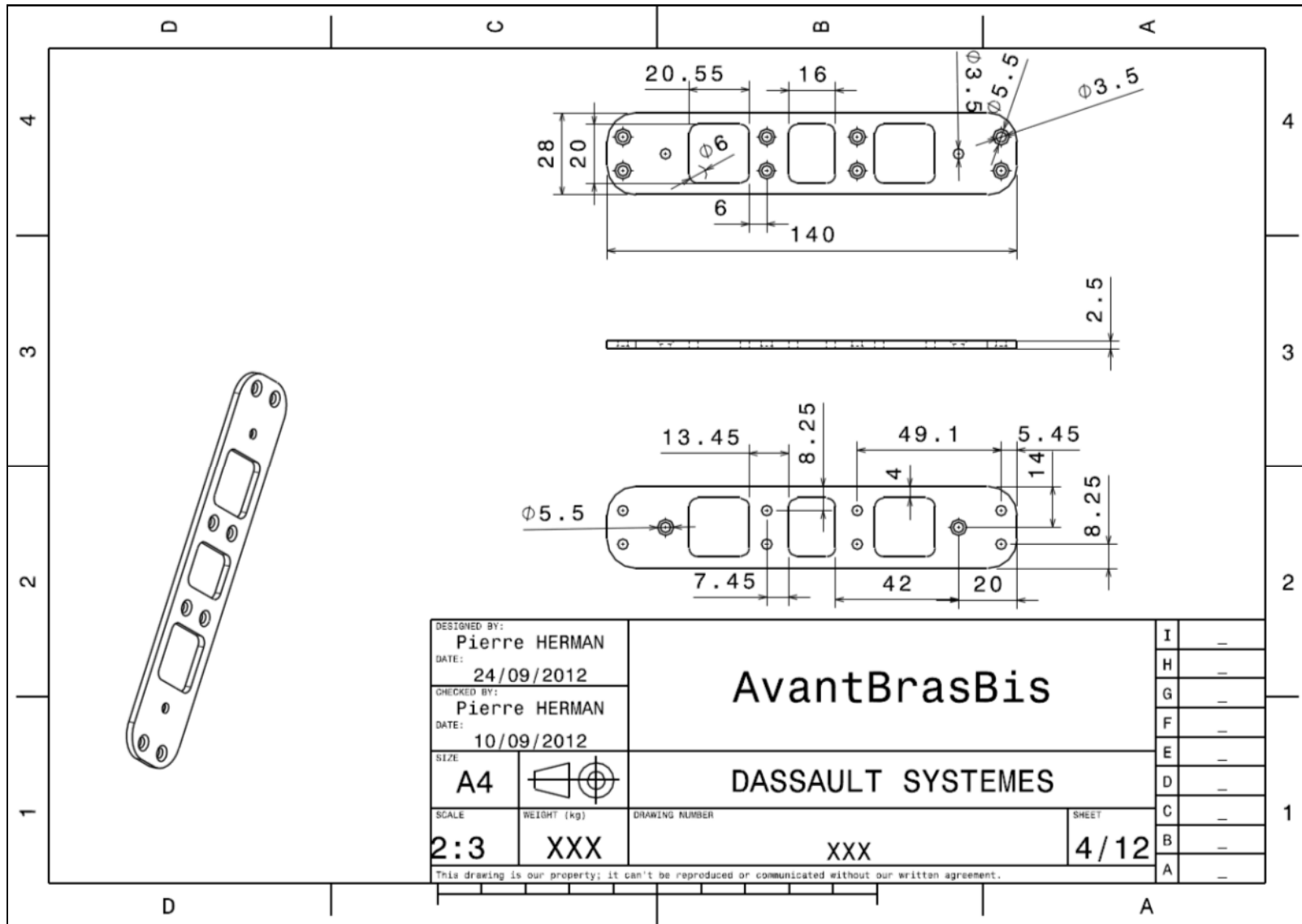
Anexo IV: Planos de las piezas

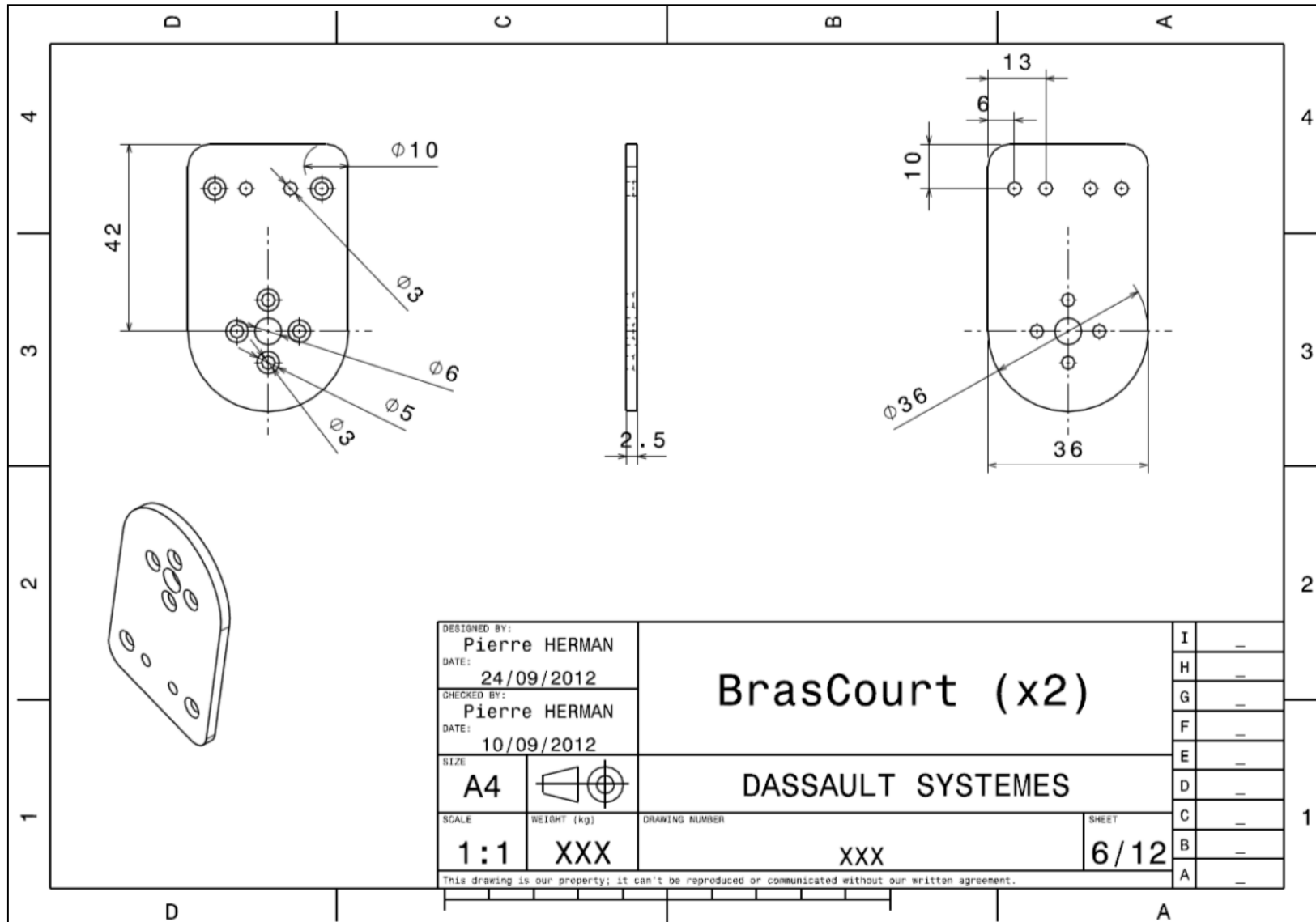


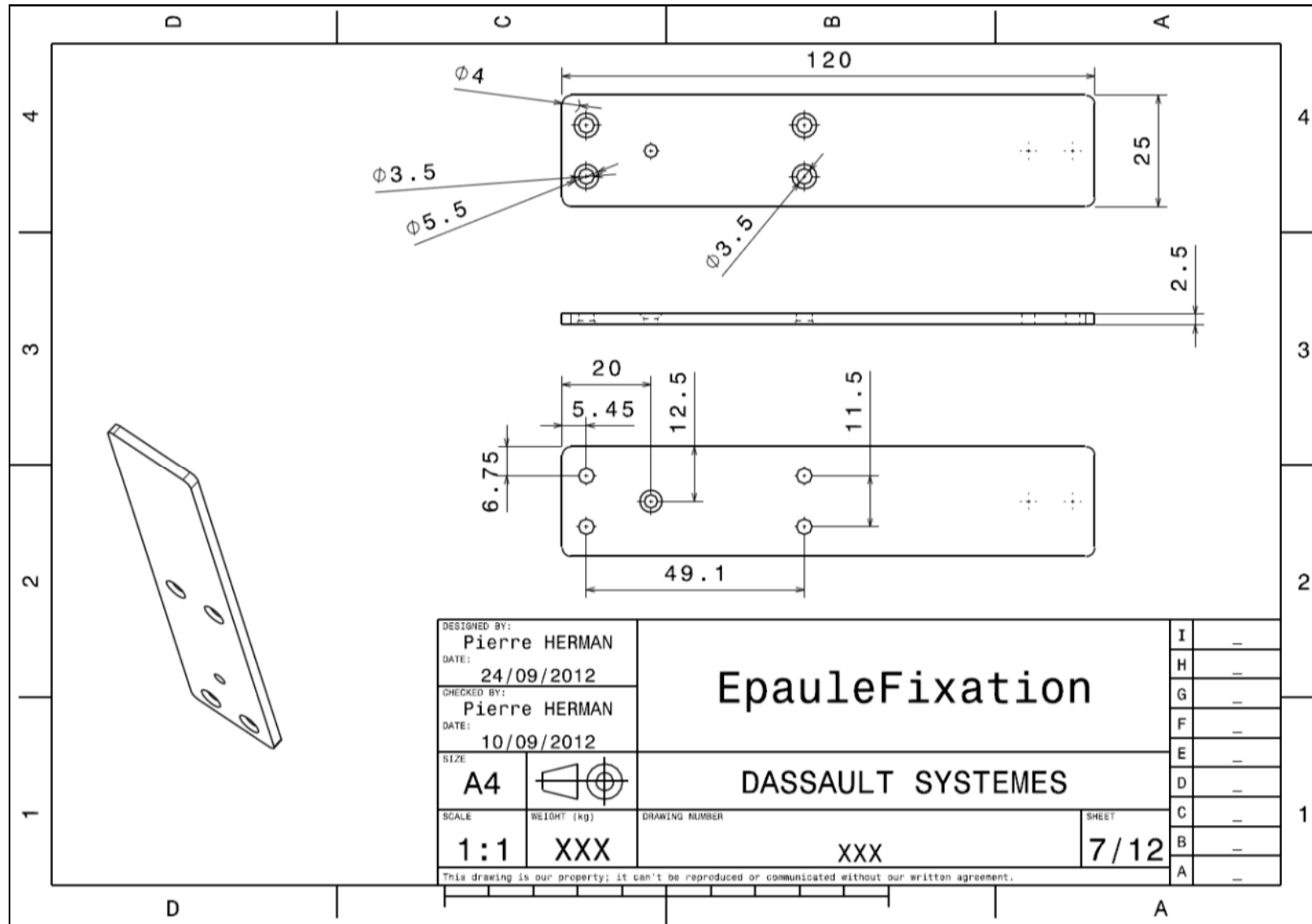


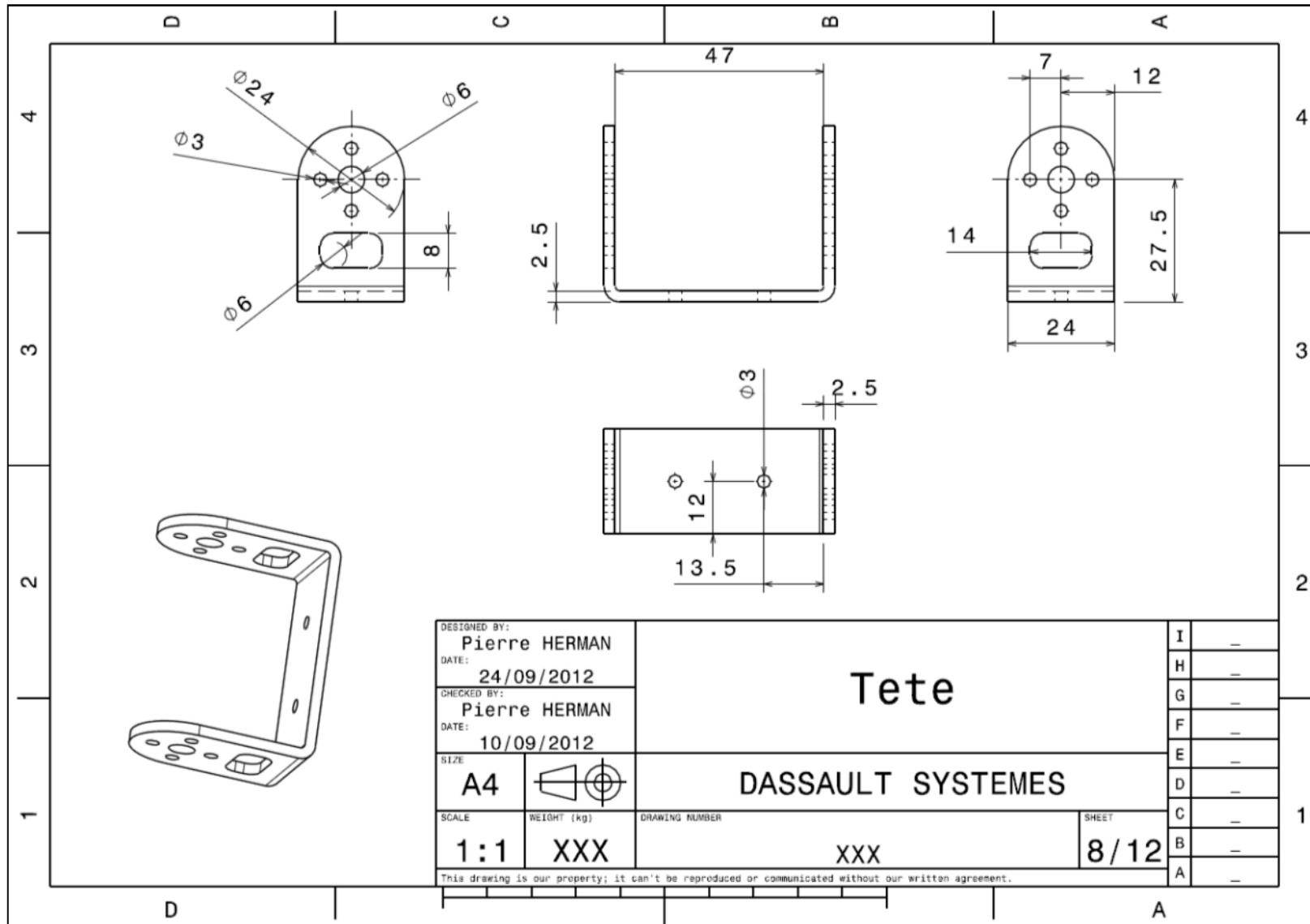


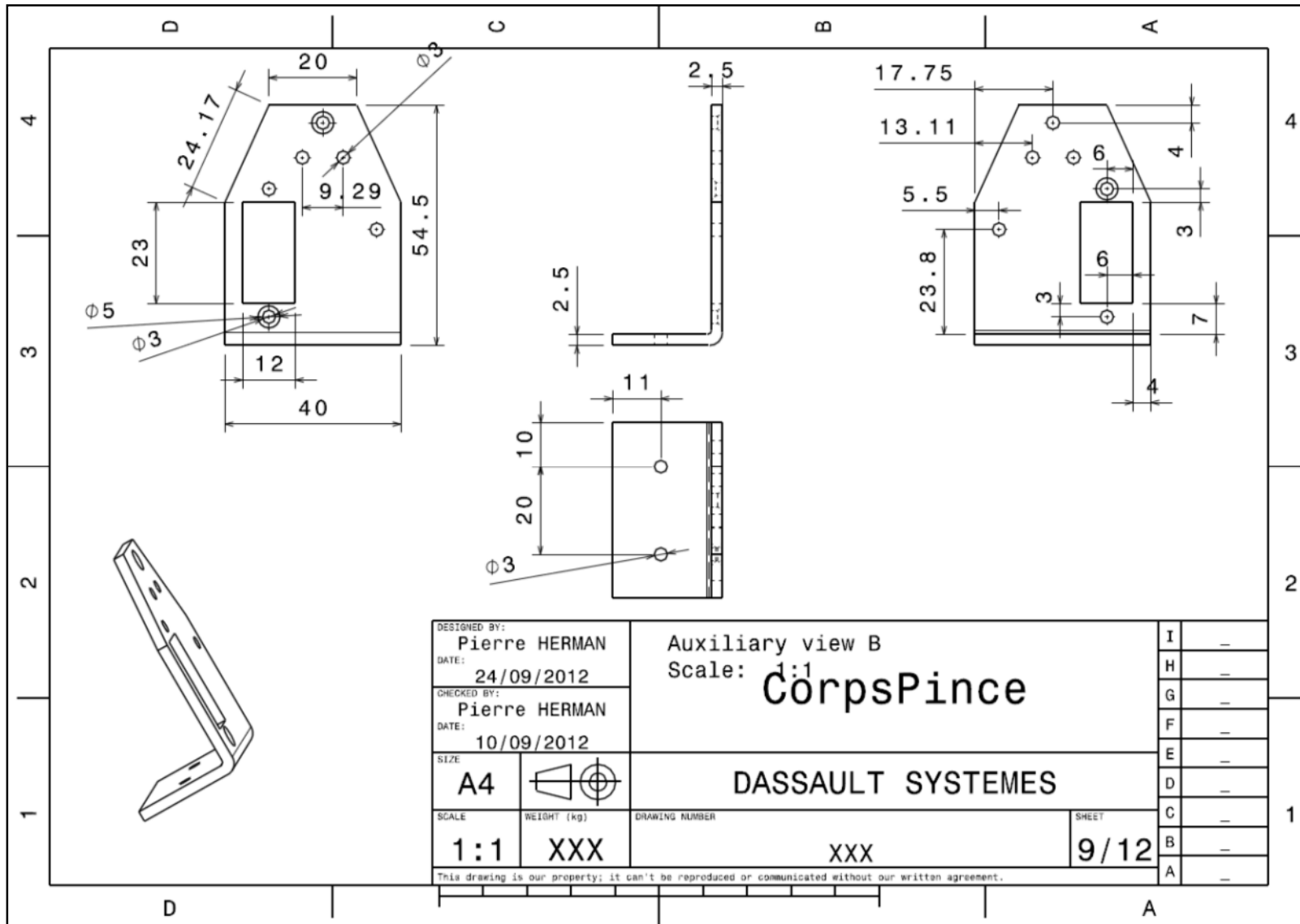


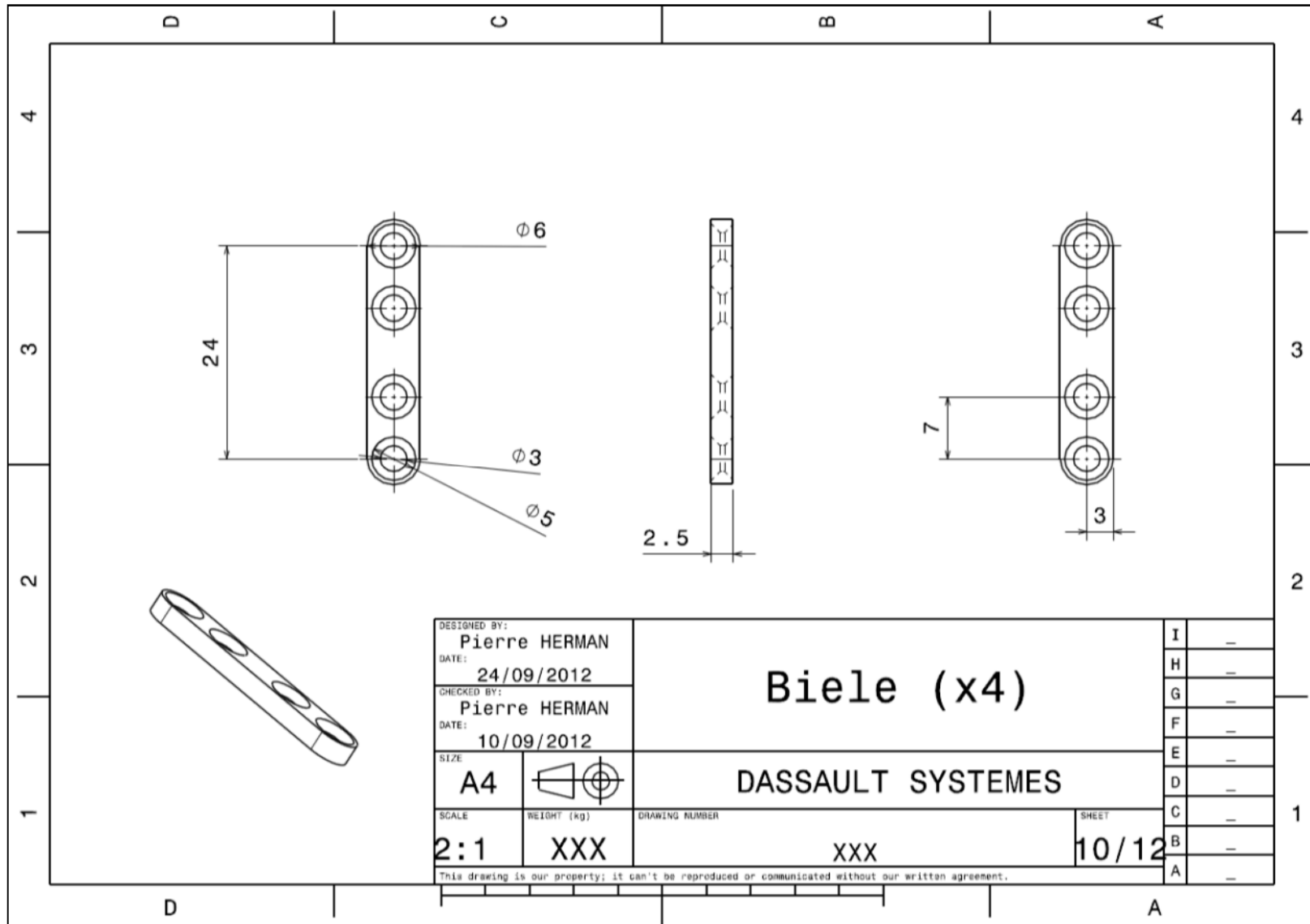


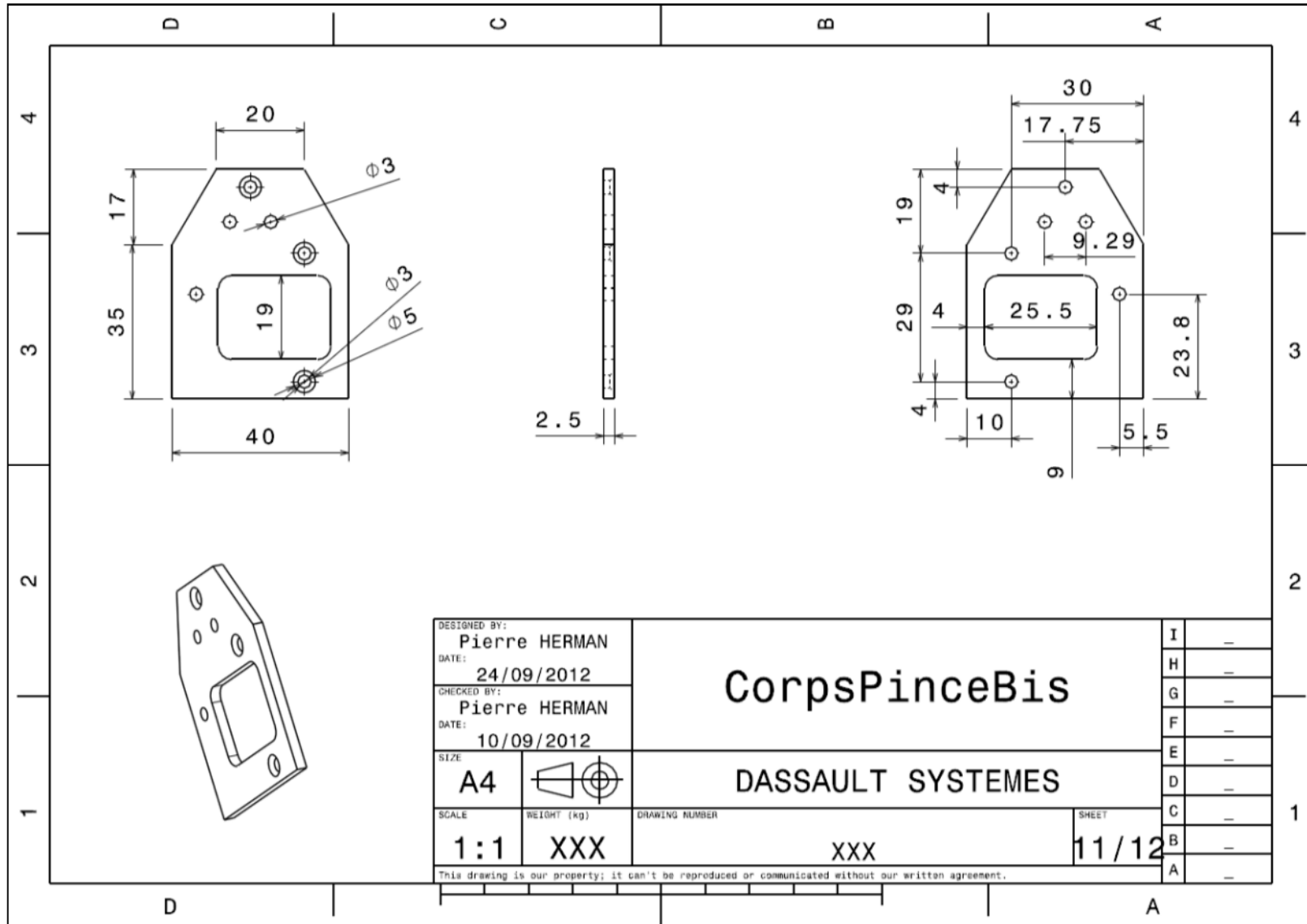


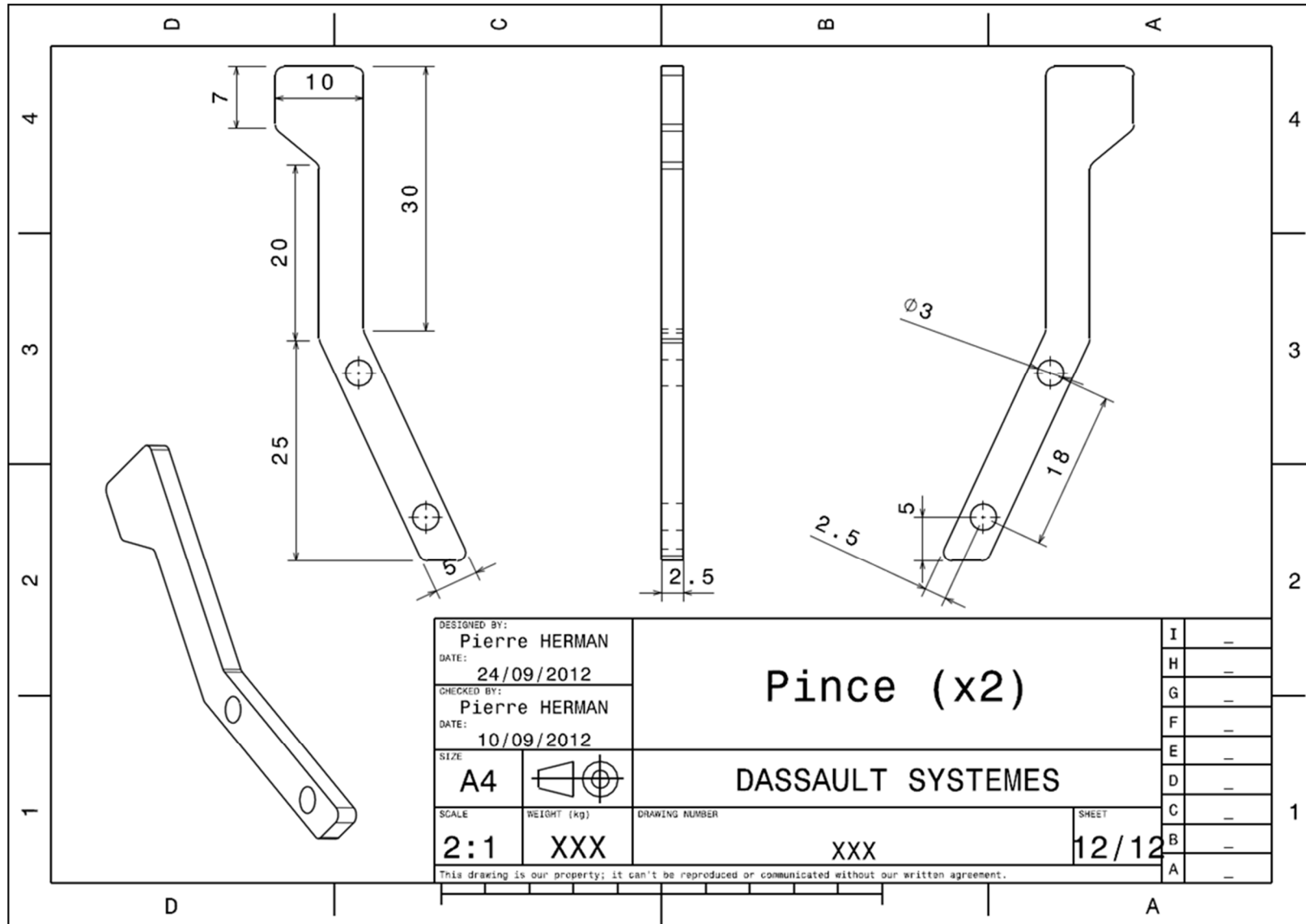






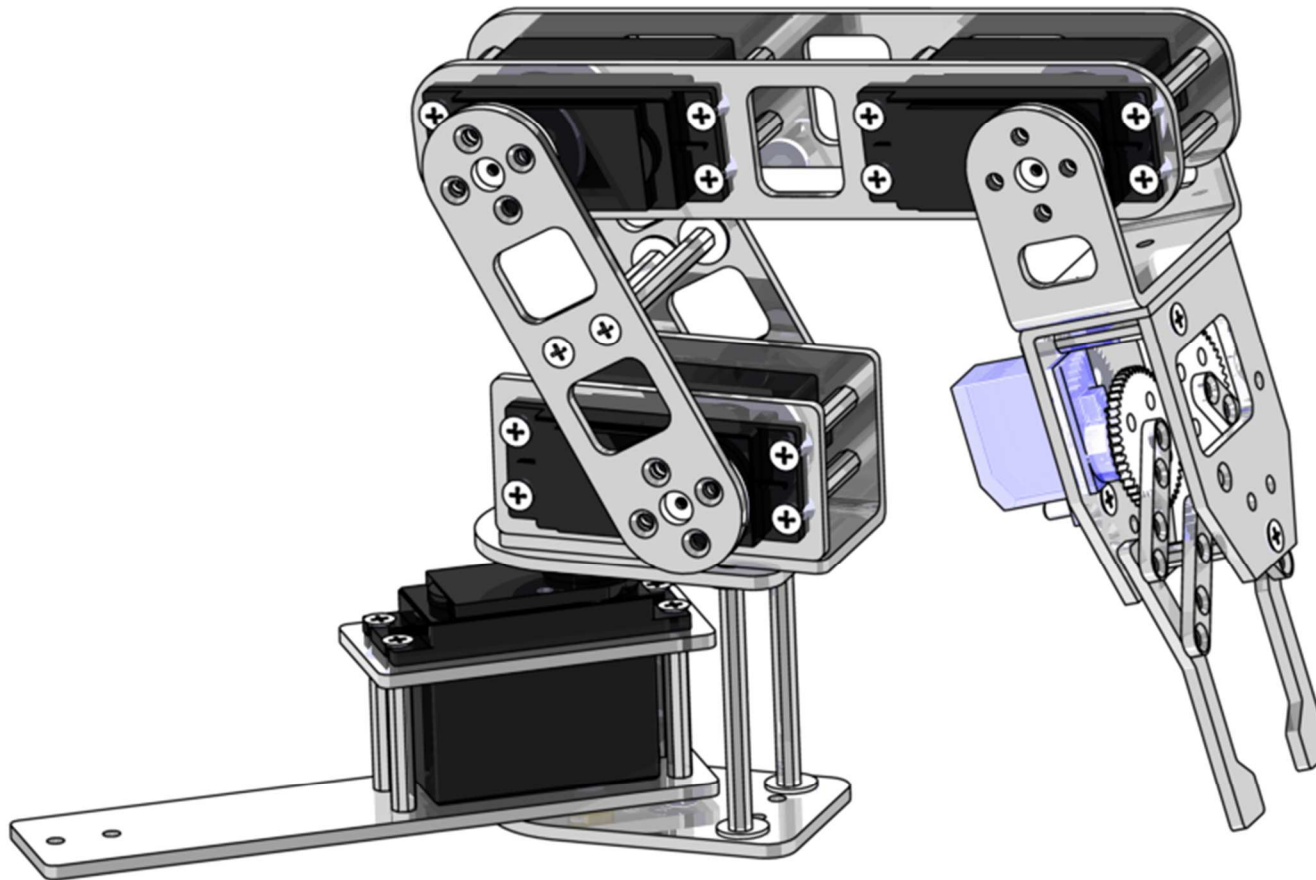


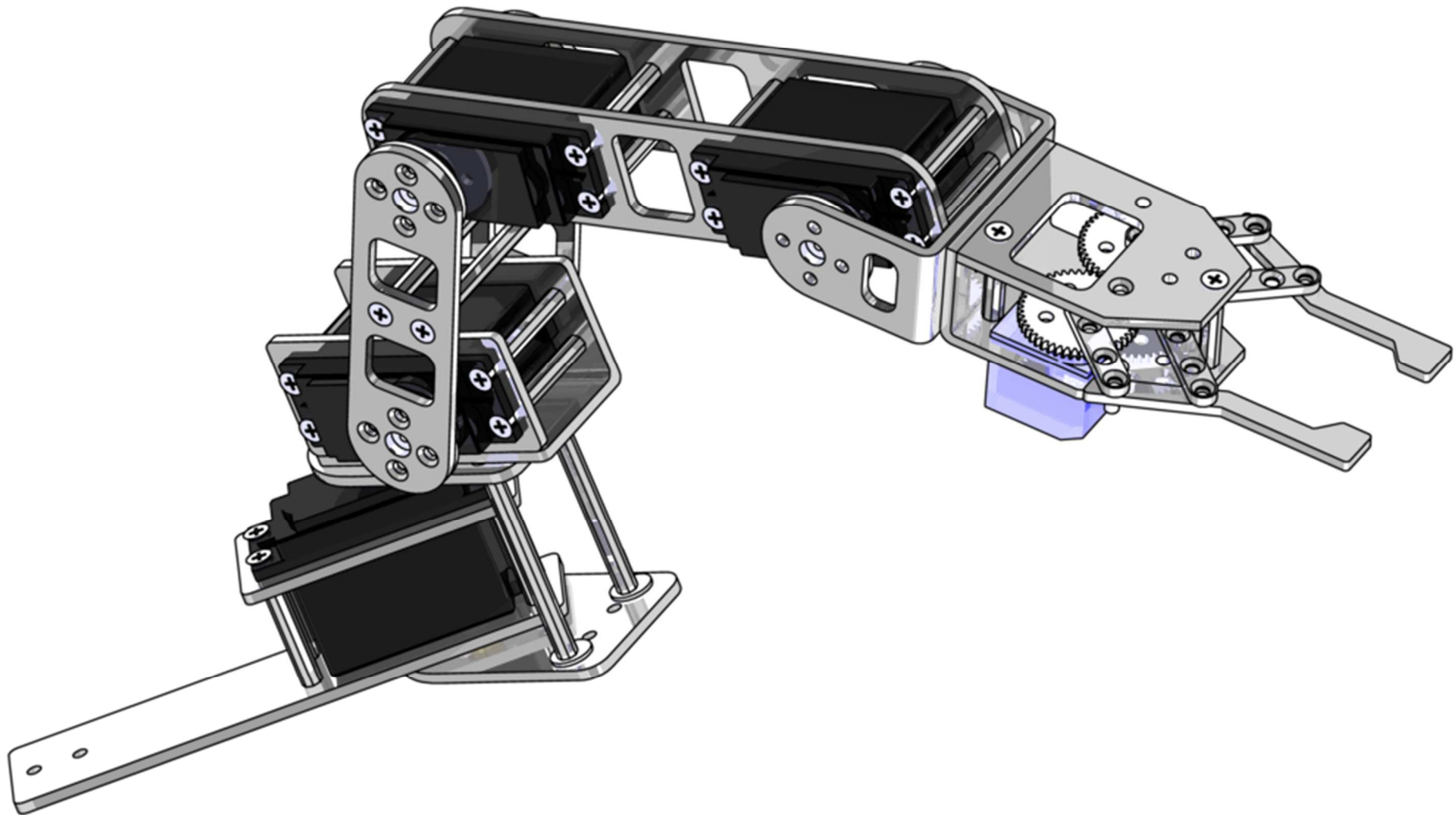


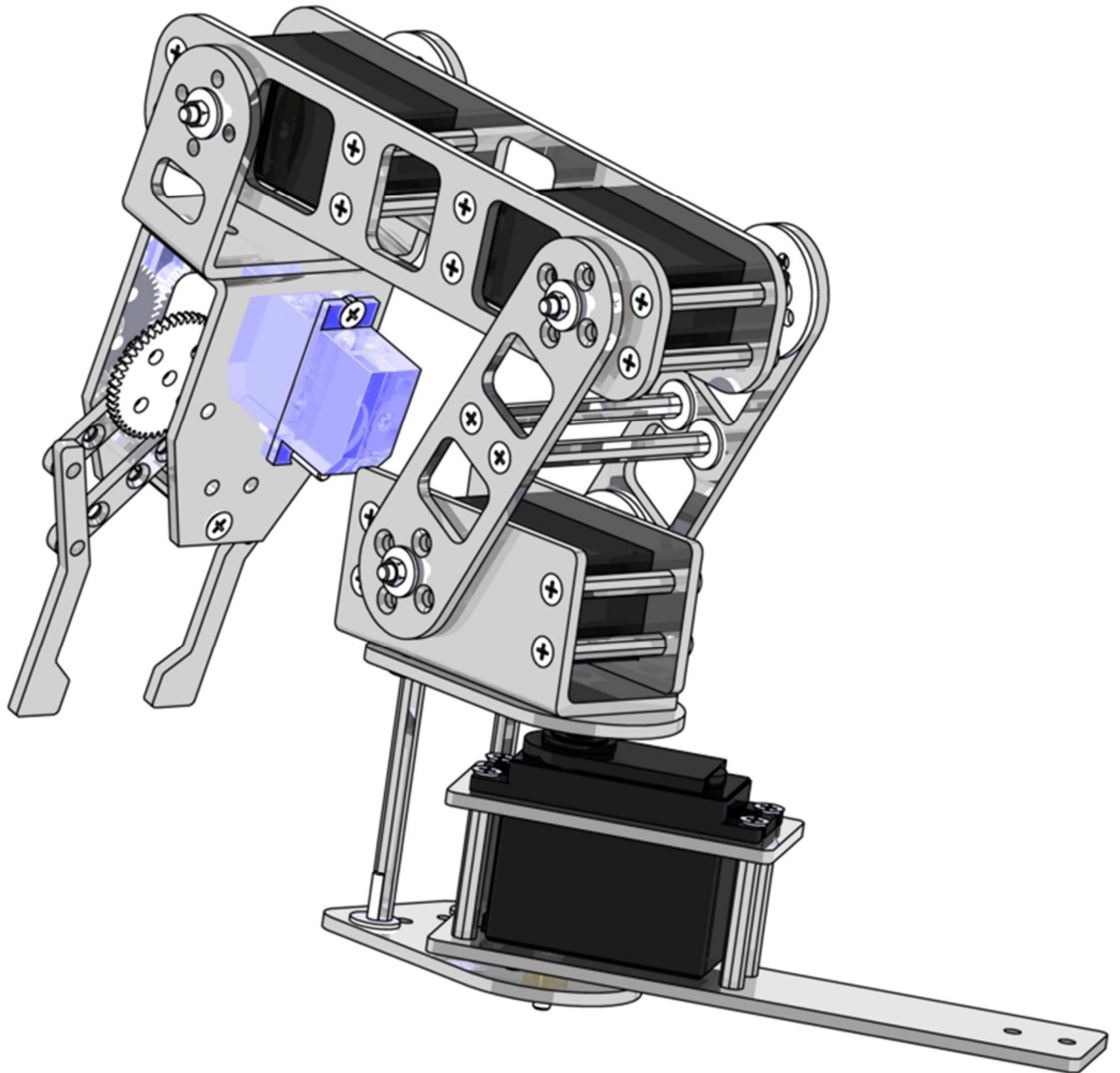




Anexo V: Renderizados del brazo robotizado



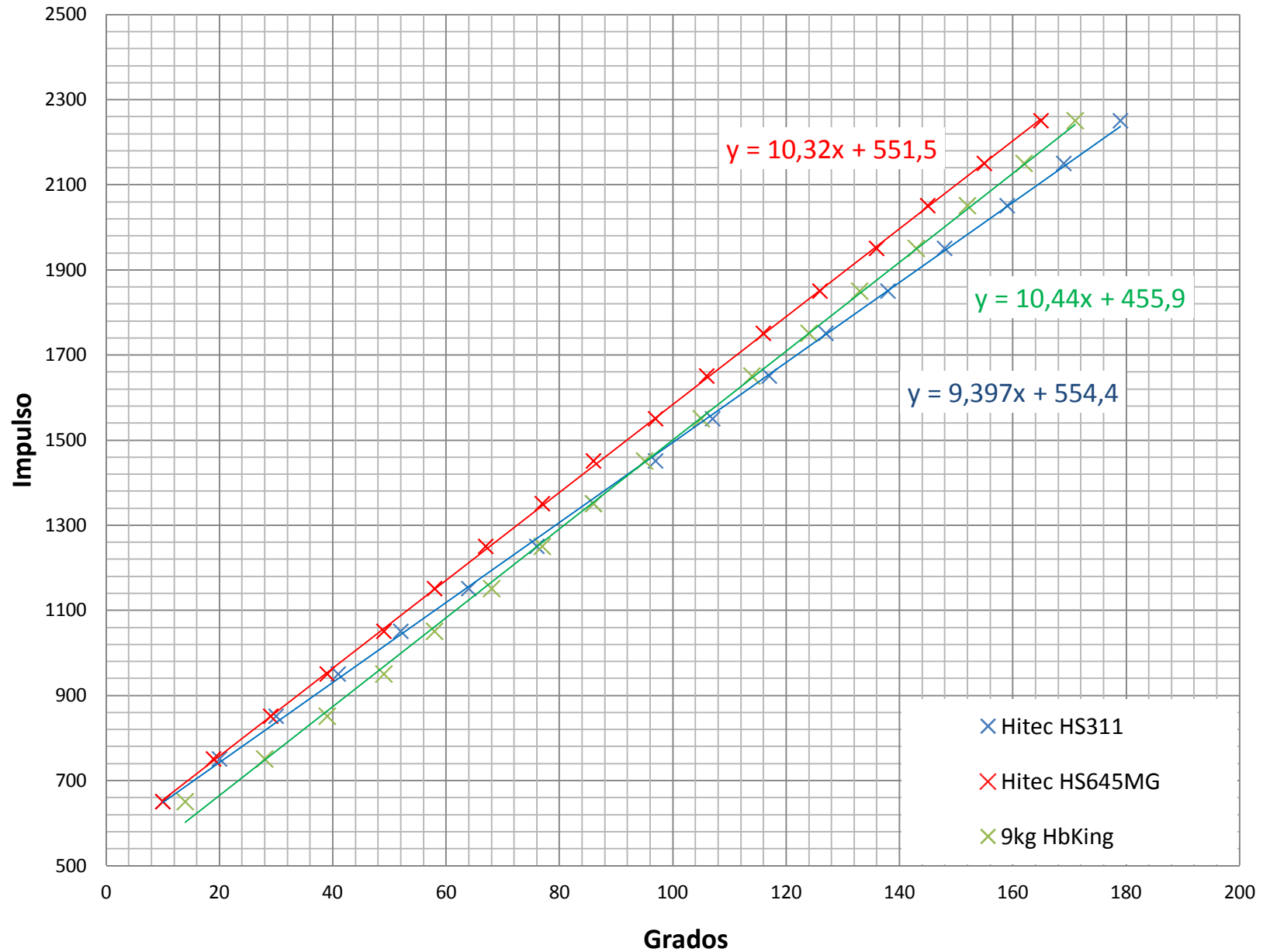


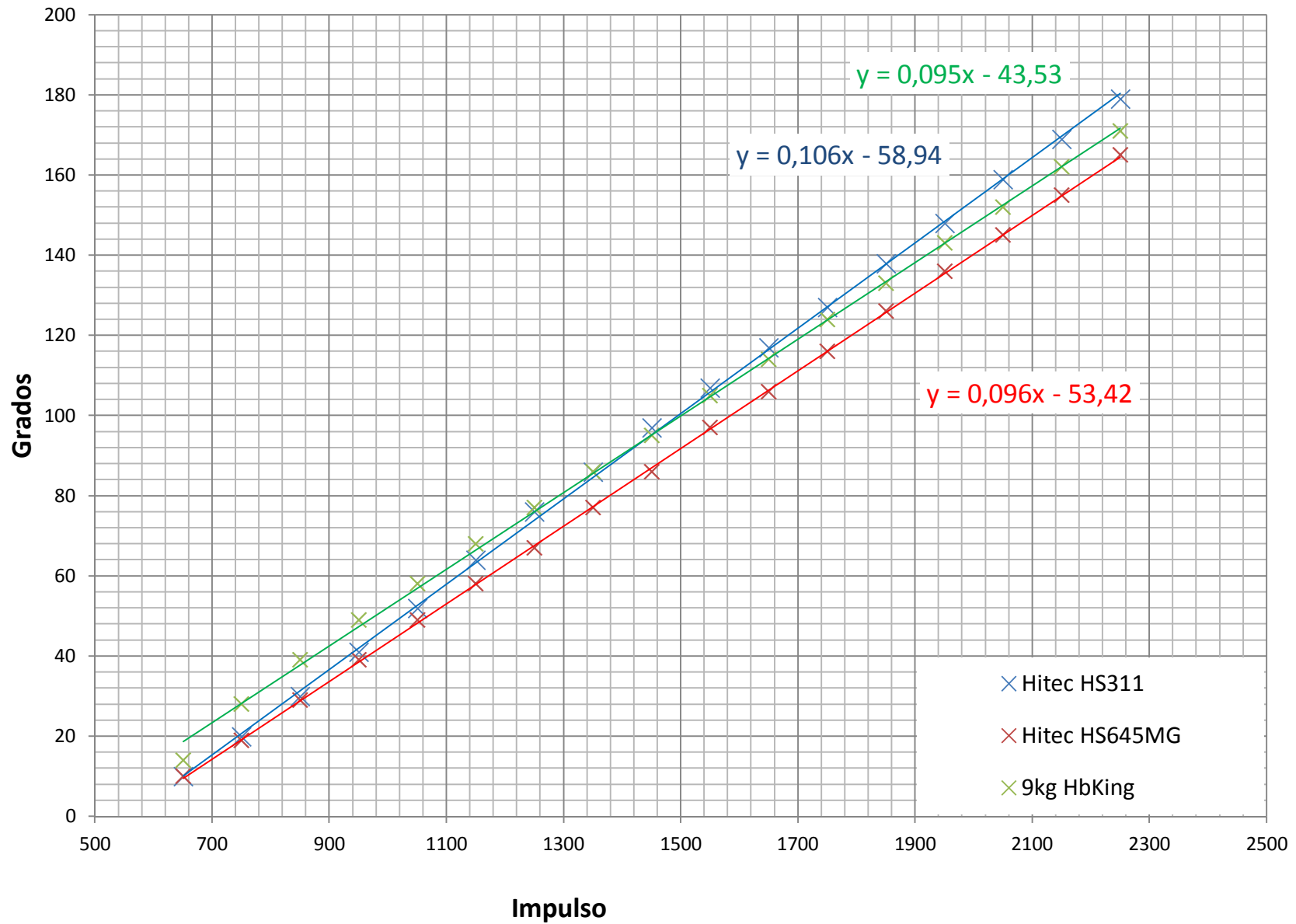




Anexo VI: Resultados de la calibración

	Grados		
Impulsión	Hitec HS311	Hitec HS645MG	9kg HbKing
650	10	10	14
750	20	19	28
850	30	29	39
950	41	39	49
1050	52	49	58
1150	64	58	68
1250	76	67	77
1350	86	77	86
1450	97	86	95
1550	107	97	105
1650	117	106	114
1750	127	116	124
1850	138	126	133
1950	148	136	143
2050	159	145	152
2150	169	155	162
2250	179	165	171





Anexo VII: Especificaciones técnicas de la Kinect



Sensor

- Lentes con sensor de color y profundidad.
- Micrófono con voz.
- Motor de inclinación para ajuste del sensor.

Campo de visión

- Campo de visión horizontal: 57 grados.
- Campo de visión vertical: 43 grados.
- Rango físico de inclinación: + - 27 grados
- Rango del sensor de profundidad: 1.2m - 3.5m.

Transmisión de datos

- 320x240 16-bit de profundidad @ 30 fotogramas/seg.
- 640x480 32-bit de color @ 30 fotogramas/seg.
- 16-bit audio @ 16 kHz.

Sistema de rastreo de esqueletos

- Rastrea hasta seis personas, incluyendo a dos jugadores activos.
- Rastrea 20 articulaciones por jugador activo.
- Habilidad para rastrear a jugadores activos, hacia los Avatares de Xbox LIVE.

Sistema de audio

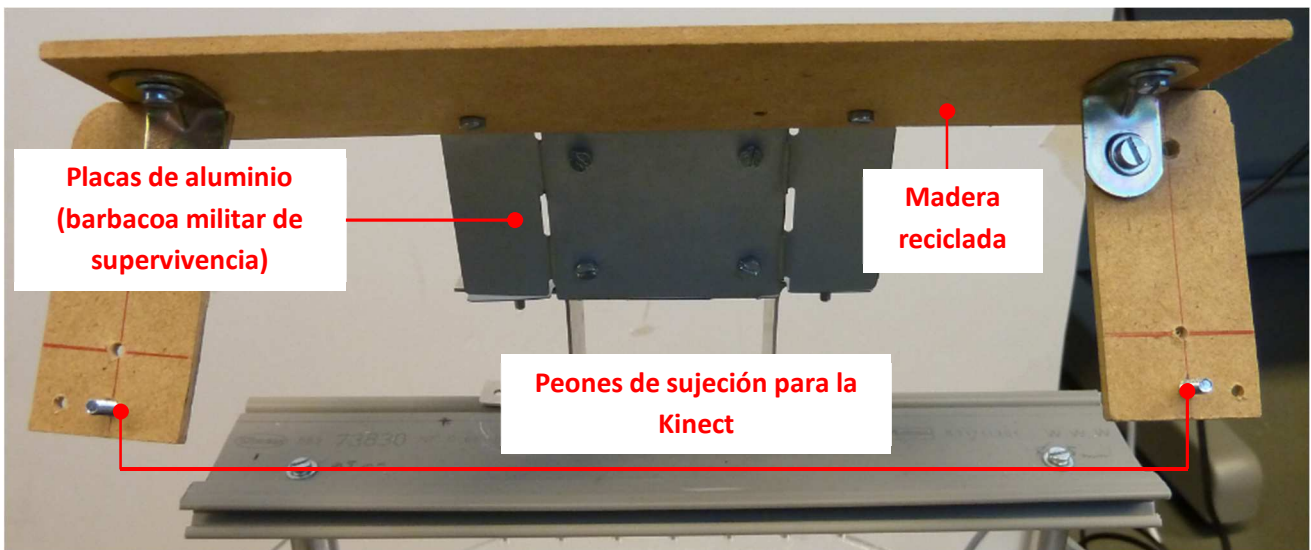
- Charla de grupo y con voz dentro de los juegos (requiere membresía Gold de Xbox LIVE).
- Sistema de cancelación de eco para la recepción de voz.
- Reconocimiento de voz en varios lenguajes.

Anexo VIII: Fabricación de piezas “artesanales”

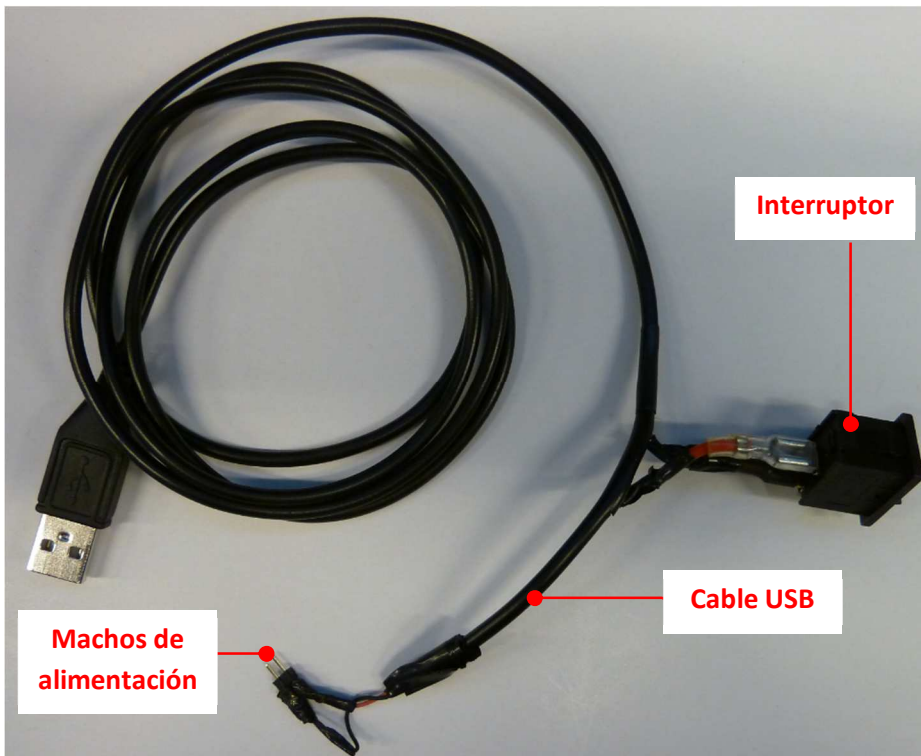
Algunas piezas necesitaron ser fabricadas de manera “artesanal”, como el soporte de la Kinect, el cable de transmisión de alimentación de los servos o el soporte de las piezas.

El soporte de la Kinect fue construido con restos de chapa de madera, escuadras, tornillos, tuercas y chapas de aluminio (mini-barbacoa de las fuerza armadas españolas). Está construido de tal manera que:

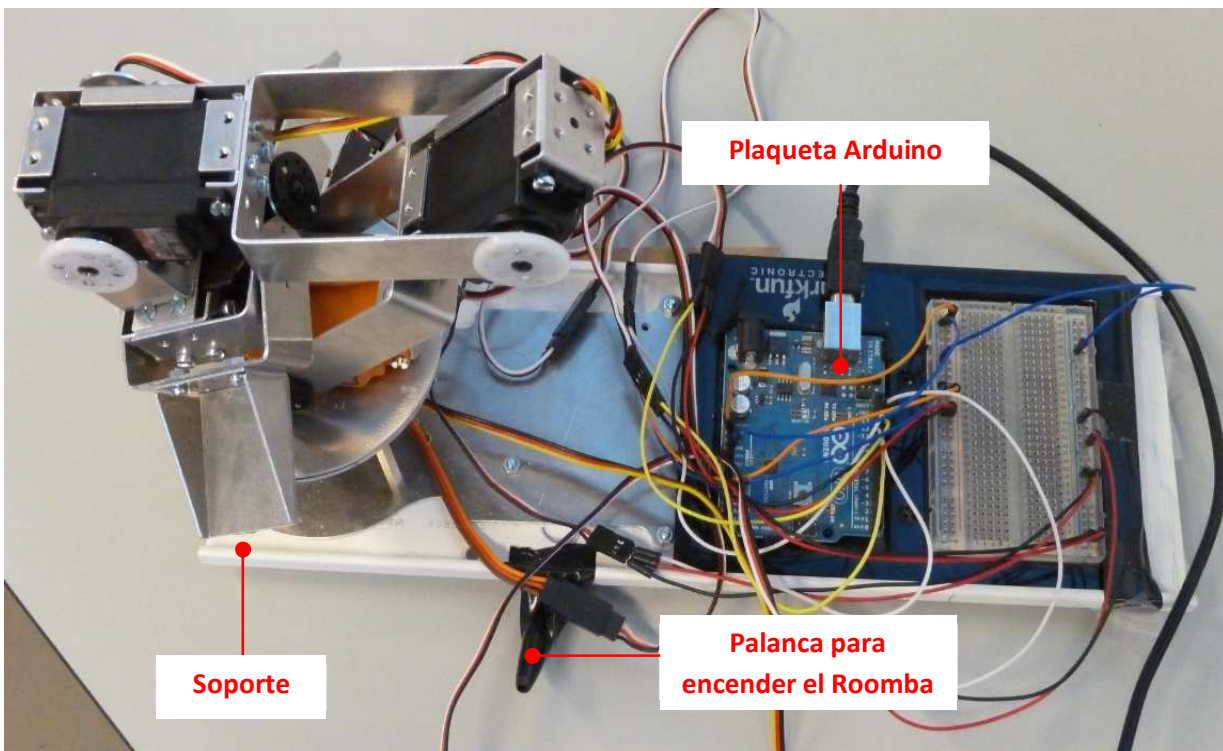
- Permite una inclinación modificable de la Kinect.
- Utiliza solamente piezas recicladas.
- Permite instalar y quitar la Kinect de manera sencilla



El cable de alimentación está hecho a partir de un cable USB estropeado y lleva un interruptor (de una vieja lámpara), así podemos cortar la alimentación de los servos sin desenchufar el conector USB.

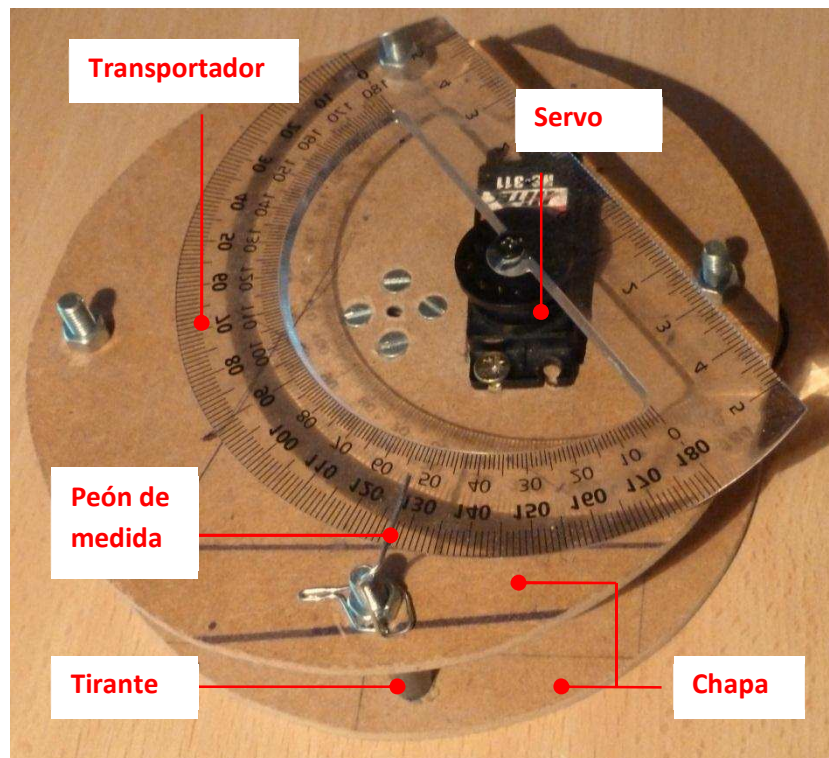


El soporte del brazo permite acoplar el brazo robotizado al robot móvil.





El calibrado permite calibrar los servos. Se reutiliza la base del robot precedente para mantener el servo. Añadimos un transportador y un peón de medida (hecho con un clip para hoja).



Anexo IX:Manual de Usuario del sistema

Manual de instalaciones:

- Instalación de Linux

Para empezar, tenemos que instalar Linux en el portátil que servirá para dirigir el robot aspiradora equipado de un brazo robotizado. Para eso, descargar el Linux Ubuntu de la red e instalarlo.

Para más detalle en la instalación de Linux, seguir el link siguiente:

<http://www.ubuntu.com/download>

- Instalación de ROS

Una vez Linux instalado, se debe instalar ROS Fuerte. Seguir las instrucciones de la página internet siguiente:

<http://www.ros.org/wiki/fuerte/Installation/Ubuntu>

- Instalación de ARDUINO

Descargar el programa e instalarlo. El programa se puede descargar a partir de la página oficial de Arduino: <http://arduino.cc/en/Main/Software>

- Instalar la biblioteca de ROS para Arduino

En el terminal de Linux, entrar instrucción siguiente:

```
>> sudo apt-get install ros-fuerte-rosserial
```

Ahora que la librería está instalada, se debe copiar el *rosserial_arduino/libraries* en el *sketchbook* de Arduino. Para eso, entrar las instrucciones siguientes en el terminal:

```
>> mkdir ~/sketchbook/sandbox/libraries/ros_lib  
>> roscd rosserial_arduino/libraries  
>> cp -r ros_lib <sketchbook>/libraries/ros_lib/
```

Para más informaciones:

http://www.ros.org/wiki/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup

Ahora copiar la carpeta *control_brazo* en el *sketchbook*.

- Modificar el fichero *ArduinoProgram.h*

Abrir el fichero *ArduinoProgram.h* en la carpeta *sketchbook/libraries/ros_lib*

Cambiar la línea `#include "WProgram.h"` por:

```
#if (ARDUINO >= 100)  
#include <Arduino.h>  
#else  
#include <WProgram.h>  
#endif
```

- Instalación del driver del Roomba

Para instalar el driver del Roomba entrar en el terminal de Linux la instrucción siguiente:

```
>> sudo apt-get install ros-fuerte-brown-drivers
```

Para más informaciones:

http://pharos.ece.utexas.edu/wiki/index.php/Writing_A_Simple_Node_that_Moves_the_iRobot_Create_Robot

- Instalar los driver de la cámara Kinect

Entrar en el terminal de Linux la instrucción siguiente:

```
>> sudo apt-get install ros-fuerte-openni-camera  
>> sudo apt-get install ros-fuerte-openni-launch
```

Para más informaciones:

http://www.ros.org/wiki/openni_camera

- Crear una carpeta de trabajo

Para empezar, entrar las instrucciones siguientes:

```
>> sudo apt-get install python-rosinstall  
>> ros_ws init ~/fuerte_workspace/opt/ros/fuerte
```

Continuar con la creación de un nuevo espacio para los futuros paquetes de instalación.

```
>> mkdir ~/fuerte_workspace/sandbox  
>> source ~/fuerte_workspace/setup.bash  
>> ros_ws set ~/fuerte_workspace/sandbox
```

- Crear un paquete con PCL para el robot

Entrar la instrucción siguiente:

```
>> source ~/fuerte_workspace/setup.bash  
>> roscd  
>> cd sandbox  
>> roscreate_pkg Nono_tutorial pcl pcl_ros roscpp sensor_msgs
```

Ahora que tenemos la carpeta de tutorial de robot, copiar en Nono_tutorial/src los ficheros "vision.cpp" "general.cpp".

En el paquete Nono_tutorial, abrir CMakeList.txt y añadir las dos siguientes líneas al final del fichero:

```
rosbuild_add_executable (vision src/vision.cpp)  
rosbuild_add_executable (general src/general.cpp)
```


Manual de arranque del robot

- Arrancar ROS

En un terminal, arrancar ROS con la instrucción:

```
>> roscore
```

- Arrancar la Cámara

Conectar el cable USB de la Kinect así que su cable de alimentación:

```
>> roslaunch openni_launch openni.launch
```

- Arrancar el programa de Arduino:

Abrir el software de Arduino. En file/Sketchbook, buscar el programa control_brazo.

Conectar el cable USB de datos y el cable de alimentación.

Enviar el programa en la plaqueta Arduino. Una vez hecho, en un terminal copiar la instrucción siguiente:

```
>> rosruntime exec rosrun rosserial_python serial_node.py /dev/ttyACM0
```

- Arrancar el nodo del Roomba:

En un terminal:

```
>> rosruntime exec roslaunch irobot_create_2_1 driver.py
```

- Arrancar el robot:

Compilar los programas en un terminal:

```
>> source ~/fuerte_workspace/setup.bash
```

```
>> roscd
```

```
>> cd sandbox
```

```
>> rosmake Nono_tutorial
```

Lanzar el programa de visión:

```
>> rosruntime exec roslaunch Nono_tutorial visión
```

Lanzar el programa general en otro terminal:

```
>> source ~/fuerte_workspace/setup.bash
```

```
>> roscd
```

```
>> cd sandbox
```

```
>> rosruntime exec roslaunch Nono_tutorial general
```