



ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECAÁNICO

EUROBOT 2016

Autor: Candelaria Utrilla Bustamante
Director: Nicolas Boullis

Madrid
Julio 2016

UNIVERSIDAD PONTIFICIA DE COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECÁNICO – PROYECTO DE FIN DE GRADO

AUTORIZACIÓN PARA LA DIGITALIZACIÓN, DEPÓSITO Y DIVULGACIÓN EN RED DE PROYECTOS FIN DE GRADO, FIN DE MÁSTER, TESIS O MEMORIAS DE BACHILLERATO

1º. Declaración de la autoría y acreditación de la misma.

El autor Dña. Candelaria Utrilla Bustamante DECLARA ser el titular de los derechos de propiedad intelectual de la obra: *Eurobot 2016*, que ésta es una obra original, y que ostenta la condición de autor en el sentido que otorga la Ley de Propiedad Intelectual.

2º. Objeto y fines de la cesión.

Con el fin de dar la máxima difusión a la obra citada a través del Repositorio institucional de la Universidad, el autor **CEDE** a la Universidad Pontificia Comillas, de forma gratuita y no exclusiva, por el máximo plazo legal y con ámbito universal, los derechos de digitalización, de archivo, de reproducción, de distribución y de comunicación pública, incluido el derecho de puesta a disposición electrónica, tal y como se describen en la Ley de Propiedad Intelectual. El derecho de transformación se cede a los únicos efectos de lo dispuesto en la letra a) del apartado siguiente.

3º. Condiciones de la cesión y acceso

Sin perjuicio de la titularidad de la obra, que sigue correspondiendo a su autor, la cesión de derechos contemplada en esta licencia habilita para:

- a) Transformarla con el fin de adaptarla a cualquier tecnología que permita incorporarla a internet y hacerla accesible; incorporar metadatos para realizar el registro de la obra e incorporar “marcas de agua” o cualquier otro sistema de seguridad o de protección.
- b) Reproducirla en un soporte digital para su incorporación a una base de datos electrónica, incluyendo el derecho de reproducir y almacenar la obra en servidores, a los efectos de garantizar su seguridad, conservación y preservar el formato.
- c) Comunicarla, por defecto, a través de un archivo institucional abierto, accesible de modo libre y gratuito a través de internet.
- d) Cualquier otra forma de acceso (restringido, embargado, cerrado) deberá solicitarse expresamente y obedecer a causas justificadas.
- e) Asignar por defecto a estos trabajos una licencia Creative Commons.
- f) Asignar por defecto a estos trabajos un HANDLE (URL *persistente*).

4º. Derechos del autor.

El autor, en tanto que titular de una obra tiene derecho a:

- a) Que la Universidad identifique claramente su nombre como autor de la misma
- b) Comunicar y dar publicidad a la obra en la versión que ceda y en otras posteriores a través de cualquier medio.
- c) Solicitar la retirada de la obra del repositorio por causa justificada.
- d) Recibir notificación fehaciente de cualquier reclamación que puedan formular terceras personas en relación con la obra y, en particular, de reclamaciones relativas a los derechos de propiedad intelectual sobre ella.

5º. Deberes del autor.

El autor se compromete a:

- a) Garantizar que el compromiso que adquiere mediante el presente escrito no infringe ningún derecho de terceros, ya sean de propiedad industrial, intelectual o cualquier otro.
- b) Garantizar que el contenido de las obras no atenta contra los derechos al honor, a la intimidad y a la imagen de terceros.
- c) Asumir toda reclamación o responsabilidad, incluyendo las indemnizaciones por daños, que pudieran ejercitarse contra la Universidad por terceros que vieran infringidos sus derechos e intereses a causa de la cesión.
- d) Asumir la responsabilidad en el caso de que las instituciones fueran condenadas por infracción

UNIVERSIDAD PONTIFICIA DE COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECAÁNICO – PROYECTO DE FIN DE GRADO

de derechos derivada de las obras objeto de la cesión.

6º. Fines y funcionamiento del Repositorio Institucional.

La obra se pondrá a disposición de los usuarios para que hagan de ella un uso justo y respetuoso con los derechos del autor, según lo permitido por la legislación aplicable, y con fines de estudio, investigación, o cualquier otro fin lícito. Con dicha finalidad, la Universidad asume los siguientes deberes y se reserva las siguientes facultades:

- La Universidad informará a los usuarios del archivo sobre los usos permitidos, y no garantiza ni asume responsabilidad alguna por otras formas en que los usuarios hagan un uso posterior de las obras no conforme con la legislación vigente. El uso posterior, más allá de la copia privada, requerirá que se cite la fuente y se reconozca la autoría, que no se obtenga beneficio comercial, y que no se realicen obras derivadas.
- La Universidad no revisará el contenido de las obras, que en todo caso permanecerá bajo la responsabilidad exclusiva del autor y no estará obligada a ejercitar acciones legales en nombre del autor en el supuesto de infracciones a derechos de propiedad intelectual derivados del depósito y archivo de las obras. El autor renuncia a cualquier reclamación frente a la Universidad por las formas no ajustadas a la legislación vigente en que los usuarios hagan uso de las obras.
- La Universidad adoptará las medidas necesarias para la preservación de la obra en un futuro.
- La Universidad se reserva la facultad de retirar la obra, previa notificación al autor, en supuestos suficientemente justificados, o en caso de reclamaciones de terceros.

Madrid, a 17 de julio de 2016

ACEPTA



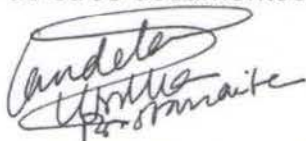
Fdo... Candelaria Utrilla Bustamante

Motivos para solicitar el acceso restringido, cerrado o embargado del trabajo en el Repositorio Institucional:

UNIVERSIDAD PONTIFICIA DE COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECAÁNICO - PROYECTO DE FIN DE GRADO

Declaro, bajo mi responsabilidad, que el Proyecto presentado con el título
Eurobot 2016 en la ETS de Ingeniería - ICAI de la Universidad Pontificia
Comillas en el

curso académico 2015/2016 es de mi autoría, original e inédito y
no ha sido presentado con anterioridad a otros efectos. El Proyecto no es
plagio de otro, ni total ni parcialmente y la información que ha sido tomada
de otros documentos está debidamente referenciada.



Fdo.: Candelaria Utrilla Bustamante

Fecha: 12/ 07/ 2016

Autorizada la entrega del proyecto

EL DIRECTOR DEL PROYECTO



Fdo.: Nicolas Boullis

Fecha: 17/ 07/ 2016

Vº Bº del Coordinador de Proyectos

Fdo.: Álvaro Sánchez Miralles

Fecha://

EUROBOT 2016

Autor: Utrilla Bustamante, Candelaria.

Director: Boullis, Nicolas.

Entidad colaboradora: ICAI-Universidad Pontificia de Comillas, CentraleSupélec

RESUMEN DEL PROYECTO

1. INTRODUCCIÓN

1.1. PLANTEAMIENTO DEL PROBLEMA

El Eurobot es una competición internacional de robótica. En Francia, su vigésimo tercera edición tuvo lugar el pasado mes de mayo de 2016. El propósito de este concurso es que sus participantes creen robots autónomos, que se enfrentarán de dos en dos en partidos de 90 segundos. Cada año, el equipo organizador publica un nuevo reglamento [1], en el que se describen el terreno de juego que se utilizará y las tareas que los robots deberán realizar.

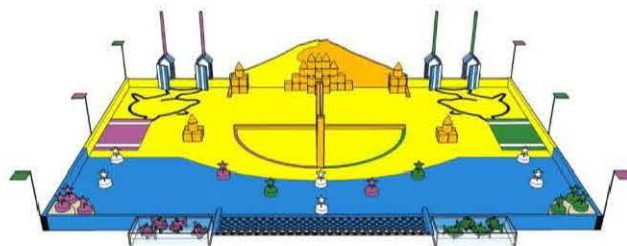


Ilustración 1. Terreno de juego de la 23ª edición del Eurobot.

El terreno de juego es un tablero simétrico ($3 \times 2 \text{ m}^2$) que contiene varios objetos en dos colores (verde o morado, cada uno corresponde a un equipo). Con éstos, los robots deben completar distintas tareas: por ejemplo, atrapar los peces de los contenedores de agua y depositarlos sobre la red.

La escuela de ingeniería CentraleSupélec cuenta con un club de robótica (el CROc) dedicado por completo a esta competición. Cada año, un grupo de alumnos integra el club y participa en el concurso, completando así un proyecto de 15 ECTS. Este año el equipo de CentraleSupélec ha emprendido la construcción de dos robots: ambos se basan en la misma arquitectura pero poseen módulos diferentes según las tareas que realizan.

El presente proyecto aborda la concepción y el desarrollo de uno de estos módulos, encargado de completar la “Recogida de Conchas”. Esta tarea consiste en recoger las conchas distribuidas por el terreno (representadas por discos de hockey). Sólo los discos pertenecientes al equipo (morados, por ejemplo) o neutros (blancos) deben ser recogidos.

1.2. ESPECIFICACIONES DEL SISTEMA

Para resolver el problema anterior, se definieron diversas especificaciones para el sistema:

- S1. Debe poder recoger y transportar los discos a la zona de partida del equipo.
- S2. Debe detectar si un disco ha sido recogido correctamente.
- S3. Sólo puede tomar aquellos discos que pertenezcan al equipo o sean neutros.
- S4. Debe poder moverse verticalmente para recoger los discos que se encuentren sobre plataformas elevadas.

Otra de las tareas a realizar por los robots era abrir una sombrilla al final del partido, empujando su eje. Sabiendo que este módulo contendría un ascensor para satisfacer S4, se definió una última especificación:

- S5. Tras 90 segundos de partido, el ascensor del sistema debe empujar el eje de una sombrilla para abrirla. Este eje no puede ser movido antes de este instante.

1.3. ESTADO DE LA TÉCNICA Y SOLUCIONES ESCOGIDAS

Una vez que se definieron los requisitos anteriores, se estudiaron las distintas tecnologías que podían ser implementadas.

S1. Con el fin de satisfacer S1, se construiría un sistema de agarre. Las pinzas por vacío y de succión fueron rápidamente descartadas, debido a su complejidad. La mejor alternativa era una pinza de sujeción mecánica.

En lo que se refiere al accionamiento de esta pinza, se optó por un servomotor, por sus múltiples ventajas (p.ej. son fácilmente controlables y realizan rotaciones con precisión).

S4, S5. Para construir un ascensor, era necesario transformar una rotación, generada por un motor, en un movimiento rectilíneo. Distintas soluciones fueron estudiadas (como las poleas o los engranajes piñón cremallera), entre las que se escogió el mecanismo de tornillo-tuerca. Éste permite convertir fácilmente una rotación en traslación, pero no a la inversa. En este sistema, esto significaba que el peso del ascensor no causaría su descenso.

Se decidió utilizar un motor paso a paso para impulsar el ascensor. Su gran ventaja es que pueden ser controlados en bucle abierto (contando los pasos efectuados).

S2. Uno de nuestros objetivos era detectar, en cualquier momento, si la pinza agarraba un objeto. Esto podía conseguirse midiendo la presión que ésta ejercía. Se realizó un estudio de los sensores de fuerza existentes hoy en día (como los FSR o las microceldas de carga)

Sin embargo, los directores de proyecto nos alentaron a probar otra solución, más interesante desde el punto de vista pedagógico. Midiendo la corriente que atravesaba el servomotor de la pinza, podíamos obtener una medida del par que ejercía.

S3. El robot sólo debe transportar discos de su color o neutros. Para conseguirlo, se plantearon dos alternativas.

La primera consistía en preconfigurar el robot. La disposición de los discos sobre el terreno era conocida, sólo existían cinco configuraciones posibles: éstas podían ser almacenadas en la memoria del robot y escogidas antes de cada partido. La segunda era crear un sistema de detección de colores.

Inicialmente, se decidió implementar la primera opción por precaución, pues se sabía que sería difícil terminar un sistema de detección de colores antes del concurso.

Respecto a la arquitectura general de los robots, se eligió una estructura maestro-esclavo. En cada robot, un maestro (RaspberryPi 2B) controlaría varias cartas esclavas (circuitos impresos basados en microcontroladores Atmega328p) mediante el protocolo i2c. Estos esclavos, a su vez, controlarían los distintos módulos del robot. Concretamente, en lo que respecta al módulo que aquí se presenta, se decidió utilizar dos esclavos: uno para controlar el ascensor; otro para la pinza y, en su caso, el sistema de detección de colores.

2. METODOLOGÍA

2.1. EL ASCENSOR

La primera tarea a realizar era concebir la estructura mecánica del ascensor. El sistema resultante consiste en una plataforma impulsada por un motor, por medio de una unión tornillo-tuerca. Durante su fabricación, se constató que la inclinación de la plataforma

bloqueaba el ascensor (fenómeno del *over-center lock* [2]). Esto se resolvió alargando la superficie de contacto entre las varillas lisas y la plataforma, usando tubos de aluminio.

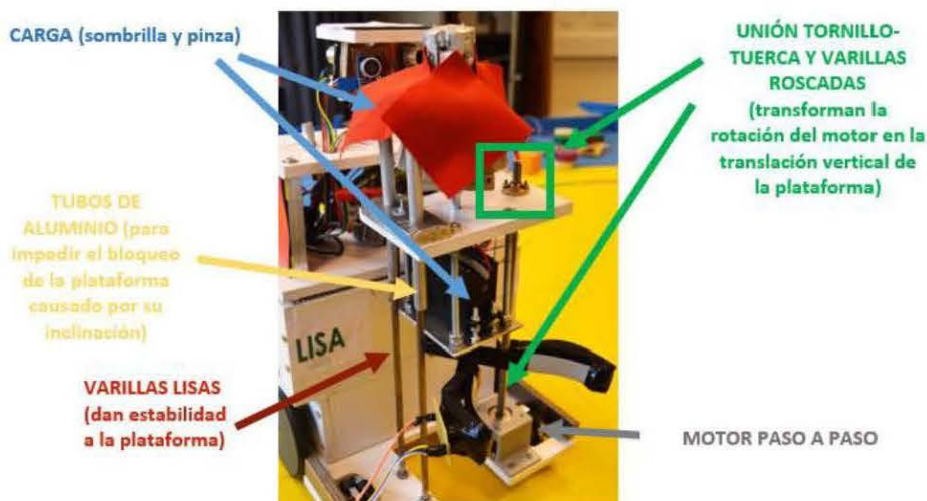


Ilustración 2. Componentes del ascensor.

Tras esto, se escogió un motor paso a paso apropiado para el sistema [3]. Para controlarlo, decidimos utilizar el driver DRV8825, muy útil al implementar una función de control de corriente. Sin embargo, crear una placa para éste controlador resultó ser una tarea particularmente difícil. Debido a su pequeño tamaño, fue imposible soldarlo. Se decidió entonces comprar un shield previamente ensamblado [4].

Por último, la carta esclava para controlar el ascensor fue creada y programada (utilizando OOP). Su programa permite al maestro gobernar el movimiento del ascensor, enviando al esclavo una orden conteniendo la posición deseada. La carta esclava se encarga de controlar la rotación del motor y, cuando el maestro lo requiere, le envía información sobre el estado del ascensor.

2.2. LA PINZA

Se construyó una pinza de sujeción mecánica de dos dedos, utilizando un mecanismo de rombo articulado (Ilustración 3). La ventaja de dicho mecanismo es que permite generar el movimiento simétrico de ambos dedos utilizando un único servomotor.

En lo que respecta a este servomotor, se escogió un Hitec-805BB. Al querer implementar un control en corriente además del control original del servomotor, en posición, se retiró el circuito electrónico del Hitec-805BB y se creó uno nuevo. En el nuevo circuito destacan tres elementos fundamentales:

- Un puente en H para controlar el motor CC.
- Un potenciómetro, para medir el ángulo de rotación del motor.
- Un subcircuito para medir la intensidad de corriente atravesando el motor. Incluye una resistencia a cuatro puntas en serie con el motor y un amplificador de instrumentación, para amplificar la tensión en bornes de dicha resistencia.

Nuestro objetivo era poder distinguir dos estados: cuando el motor funcionaba en vacío y cuando estaba bloqueado (es decir, cuando la pinza agarraba un objeto). Para ello, era necesario obtener una medida de la corriente media atravesando el motor: una señal plana que pudiese ser analizada con un microcontrolador. Sin embargo, la corriente oscilaba debido al control en PWM del motor. Además, otros ruidos tuvieron que ser mitigados

(provenientes de señales parásitas y de instrumentos de medida). Con la ayuda de varios filtros, se obtuvo una señal explotable.

Tras esto, se pudo crear y programar la segunda carta esclava. Su código sigue el mismo principio que el creado para el ascensor.

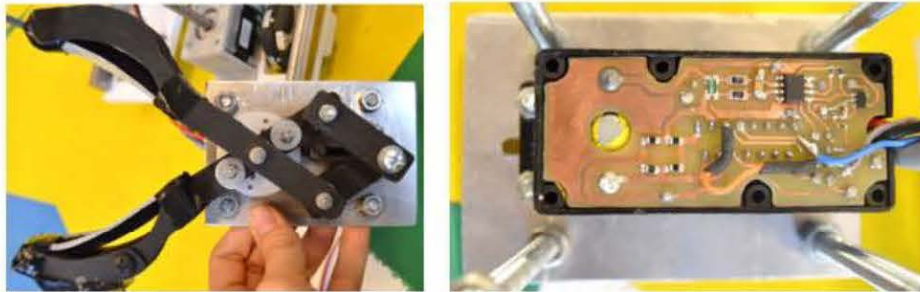


Ilustración 3. A la izquierda, se puede observar el mecanismo de la pinza (rombo articulado). A la derecha, el nuevo circuito del servomotor, integrado dentro de su carcasa.

2.3. EL SENSOR DE COLOR

Por último, se concibió un subsistema de detección de colores. Éste no pudo ser desarrollado antes del concurso, pero se decidió crearlo después por el interés pedagógico de la tarea. El componente principal del sistema es el sensor óptico TCS3200. Éste permite analizar la luz proyectada sobre él con cuatro tipos de fotodiodos (sin filtro o con filtros rojo, verde o azul) y da, en respuesta, una señal cuadrada de frecuencia proporcional a la irradiancia medida (Potencia/Superficie). El objetivo era analizar la luz reflejada por los objetos del terreno de juego y obtener información sobre el carácter rojo, azul y verde de esta luz. Posteriormente, esta información sería analizada para poder distinguir cuatro colores presentes en el terreno: blanco, verde, morado (conchas) y naranja (bloques de madera, que simbolizan arena).

Las primeras pruebas permitieron descubrir que las medidas del TCS3200 se veían perturbadas por las variaciones de luz ambiental. Por ello, se decidió crear un prototipo en el que el sensor estaría aislado y los objetos observados se iluminarían con LEDs de luz blanca. Además se tuvieron en cuenta otras consideraciones para este diseño (p.ej., que la luz de los LEDs no debe iluminar directamente el sensor). El prototipo resultante, que incluye un PCB y una carcasa, se puede observar en la Ilustración 4.

El prototipo fue probado y su funcionamiento fue el esperado. Además se pudo concluir de estos segundos tests que, como no todas las frecuencias de luz son emitidas con la misma intensidad por los LEDs blancos, las medidas del TCS3200 debían normalizarse.

Inicialmente, la frecuencia de salida de este sensor se medía con un osciloscopio. Más tarde, se creó un programa con el que medirla utilizando un Atmega328p (gracias a la función *pulseIn()* de Arduino). Tras esto, se pudo concebir un programa para identificar colores. Éste se basa en un sistema vectorial. Cada color está representado por un vector de tres componentes (ROJO; AZUL; VERDE) y varios colores conocidos están guardados en memoria. Cuando se desea medir un nuevo color, se compara con los vectores guardados en memoria (calculando distancias Euclidianas) y se identifica con el color correspondiente al vector más cercano. Este programa fue puesto a prueba con los objetos del terreno de juego y se pudieron identificar todos los colores sin problema.

Finalmente, el sensor fue integrado en el subsistema de la pinza. Para ello, se modificó el código de la carta esclava correspondiente, con el fin de incluir la nueva función de detección del color, y se creó una nueva pinza mecánica.



Ilustración 4. A la izquierda, el prototipo del sistema de detección de colores. A la derecha, el mismo prototipo instalado en la pinza.

3. RESULTADOS

Antes del concurso, se pudo obtener un módulo funcional. Tanto el subsistema de elevación como el de agarre fueron probados y funcionaban de la manera esperada. También se comprobó que la interacción de las cartas esclavas con el maestro funcionaba correctamente. Sin embargo, la falta de tiempo antes de la competición nos hizo incapaces de identificar algunos problemas, que fueron detectados durante el concurso.

En primer lugar, cuando el motor paso a paso estaba alimentado pero no era utilizado, consumía demasiada corriente, ya que los puentes en H alimentaban constantemente sus bobinas. Esto se pudo mejorar más tarde porque, gracias al sistema tornillo-tuerca, el ascensor se sostenía sin necesidad de alimentar el motor.

Por otro lado, el subsistema de agarre siempre funcionó como era esperado. Tan sólo el sistema mecánico se quedó bloqueado, lo que se solucionó engrasando la pinza.

Respecto al sistema de detección del color, dio resultados muy satisfactorios. Incluso cuando éste fue instalado en la pinza, donde la luz ambiente podía llegar al sensor (dependiendo de cómo la pinza agarrase su carga).

4. CONCLUSIONES

La concepción y construcción de este módulo ha sido una tarea muy completa, que ha permitido el desarrollo de múltiples competencias en electrónica y otras áreas de conocimiento. Este proyecto ha permitido crear un sistema integral y trabajar en cada una de las fases de su desarrollo. Es necesario remarcar la importancia de la integración de la mecánica, la electrónica y la programación: una tarea difícil, presente en cualquier proyecto industrial hoy en día. Además, se ha ganado la oportunidad de realizar tareas no técnicas pero de gran utilidad e interés (como la recaudación de fondos).

5. REFERENCIAS

- [1] Planète Sciences, “Eurobot. The Beach Bots-2016. Rules”, Septiembre 2015.
http://www.eurobot.org/attachments/article/51/E2016_Rules_EN.pdf
- [2] Noguét, “GUIDER EN TRANSLATION: la liaison glissière”, Abril 2009
<http://sitemecanique.free.fr/Documents%20TGE1/Etude%20des%20constructions/Technologie/CI6%20Analyser%20et%20concevoir%20la%20solution%20constructive%20d'une%20liaison/Liaison%20a%201%20degre%20de%20liberte/Cours%20Guidage%20en%20translation.pdf>
- [3] Pololu Robotics and Electronics, Stepper Motor with 18cm Lead Screw
<https://www.pololu.com/product/2689>
- [4] Pololu Robotics and Electronics, DRV8825 Stepper Motor Driver Carrier
<https://www.pololu.com/product/2133>

Author: Utrilla Bustamante, Candelaria.

Director: Boullis, Nicolas.

Collaborating entity: ICAI-Universidad Pontificia de Comillas, CentraleSupélec

ABSTRACT

1. INTRODUCTION

1.1. PROBLEM STATEMENT

The Eurobot is an international robotics competition. In France the twenty-third edition of Eurobot took place in May 2016. The aim of the competition is for the participants to create autonomous robots that will play against each other in two-team 90-second matches. The organizing team publishes each year a new regulation [1], describing the playing field which will be used and the tasks the robots will have to perform.

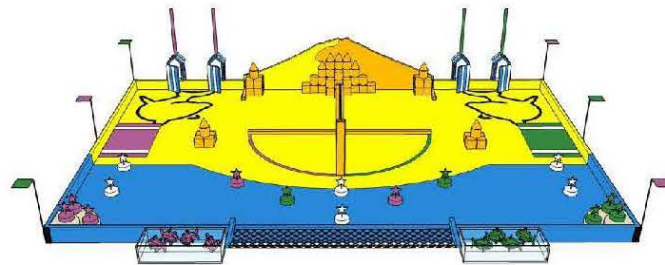


Figure 1. The Eurobot 23rd edition playing field.

The playing field contains several objects with which the robots will accomplish different tasks (e.g. catching the fish in the water tanks and placing them on the net). It is a symmetrical 3x2 m² field with objects in two colors (purple and green, each color corresponding to a team).

The engineering school CentraleSupélec robotics club (the CRoC) is entirely devoted to this competition. Each year, a group of students joins the club and they turn the participation at the Eurobot into a 15 ECTS annual project. This year, we undertook the conception of two robots: both have been built using the same architecture, but have different modules depending on the tasks they have to perform.

This report presents the conception and development of a specific module in one of the robots, in charge of performing the task “Seashell Collecting”. It consists in transporting the seashells throughout the playing field (hockey disks) to the team’s area of departure, but only the team’s ones (purple for example) and the neutral ones (white)

1.2. SYSTEM SPECIFICATIONS

In order to solve the previous problem, we defined several specifications for the system:

- F1. It has to be able to grab and transport the disks to the area of departure.
- F2. It has to detect if the system catches a disk properly.
- F3. It will only carry the team’s disks or the neutral ones.
- F4. It has to be able to move vertically to reach the disks placed on the field elevated platforms.

Another task our robots needed to accomplish was to unfold an umbrella once the match had finished, by pushing its shaft. Knowing a lift had to be built to fulfil F4, a last specification was defined:

- F5. At the end of the first 90 seconds of a match, the lift has to push an umbrella's shaft in order to deploy it. The shaft can not be moved before this moment.

1.3. STATE OF THE ART AND TECHNOLOGIES CHOSEN

Once these requirements were determined, we studied and chose the technologies that could be implemented in the module.

F1. To fulfil F1, a gripper had to be built. Technologies such as vacuum or suction grippers were early discarded, due to their complexity. The best choice was a mechanical clamp.

Concerning the actuation of the clamp, we opted for servomotors because of their many advantages (e.g. they perform rotations with accuracy and can be easily commanded).

F4, F5. In order to build an elevator, we needed to transform a motor rotation into a linear movement. Several options were studied (like pulleys or rack-and-pinion assemblies) among which we opted for a screw-nut mechanism. One advantage of this system is that a rotation can be easily transformed into a linear translation, but the opposite is very difficult. This means that the weight of the elevator would not make it fall.

To generate the rotation, we chose stepper motors. They present one great advantage: the position of their load can be controlled in open loop (by counting its steps).

F2. Our aim was to detect, at any time, if the gripper was grabbing an object, which could be achieved by measuring the pressure exerted by the clamp. A study of the existing force sensing technologies (such as FSR sensors or micro-load cells) was performed.

However, our teachers encouraged us to implement another solution, of greater pedagogical interest. By measuring the intensity flowing through the gripper's servo motor, we could obtain a measure of the torque it applied.

F3. The robot may only transport seashells of its color or neutral ones. In order to achieve it, two possibilities were discussed.

The first one was to preconfigure the robot. The way seashells would be distributed throughout the playing field could be known in advance. Indeed, there were only five different possible configurations: they could be stored in the robot's memory and chosen before each match. The second one was to create a color-sensing system.

We started by implementing the first one, as a precaution, knowing that a color-sensing system would be hard to finish before the contest.

Concerning the general architecture of our robots, we chose a master/slave structure. In each robot, a master card (RaspberryPi 2B) would control several slave cards (printed circuit boards based on Atmega328P microcontrollers) using the i2c protocol. These slave cards would, in turn, control the different modules of the robot. More specifically, it was decided that our module would include two slave cards: the first one to control the elevator; the second one for the gripper and, eventually, the color-sensing subsystem.

2. METHODOLOGY

2.1. THE ELEVATOR

The first job to undertake was the conception of the elevator mechanics. The resulting design consisted of a platform driven by a motor, using a screw-nut system. During the creation of this mechanical structure, we experienced an over-center locking phenomenon

[2] that blocked the elevator. This problem was resolved by lengthening the contact surface between the platform and the smooth rods, using aluminum tubes.

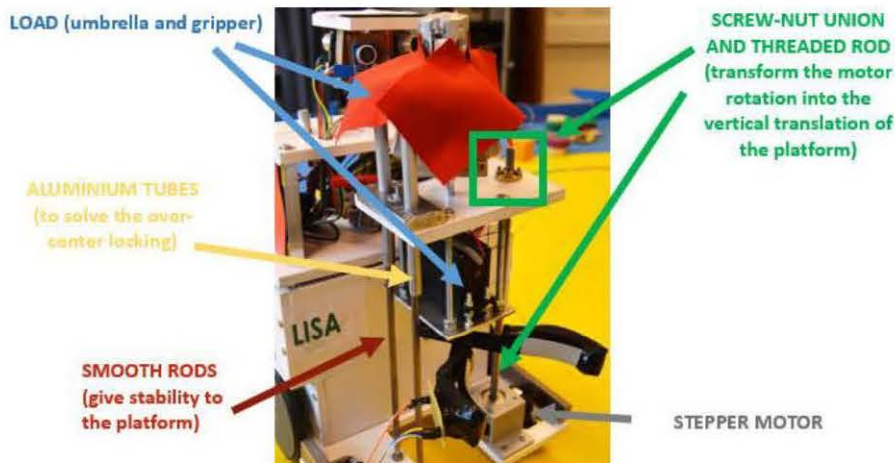


Figure 2. Finished elevator components.

Afterwards, we were able to choose an appropriate stepper motor for the system [3]. To control it, we opted for the driver DRV8825, very convenient since it implements current protection. However, creating a shield for this component happened to be a particularly difficult task. Due to the small format of the component, we were unable to weld it and to create a shield for it. We then decided to purchase an already-assembled one [4].

Finally, we created the slave card which controls the elevator and programmed it (using OOP). The program allows the master to control the elevator by sending an order containing the target position. The slave commands the rotation of the motor and, when requested by the master, sends information concerning the state of the elevator.

2.2. THE GRIPPER

With respect to the gripper mechanics, we chose to build a two-finger clamp using a rhomboid mechanism (Figure 3). The advantage of this system was that it allowed to generate the symmetrical movement of both fingers using a single servo motor.

Concerning the gripper's actuator, a servo motor Hitec-805BB was chosen. Since we wanted to implement a current control in addition to the original position control of the Hitec-805BB, we soon decided to remove its electronic circuit to reconceive it ourselves. The new circuit can be summarized in three essential elements:

- An H-bridge to control the DC motor.
- A potentiometer, to measure the angle of rotation of the rotor.
- A sub-circuit to measure the current flowing through the motor. It includes a kelvin resistor in series with the motor and an instrumental amplifier, to amplify the voltage between the terminals of this resistor.

Our aim was to distinguish two states: when the motor was running on empty and when it was blocked (i.e. when the gripper was grabbing an object). For that, we needed a measure of the average current through the motor: a flat signal that would be analyzed with a microcontroller. However, this current oscillated due to PWM motor control. In addition, we had to overcome other disruptions caused by parasitic signals and by the measuring instruments. Finally, using several filters, we obtained an exploitable measure.

The second slave card was then created and programmed. Its code follows the same principle as the elevator code.

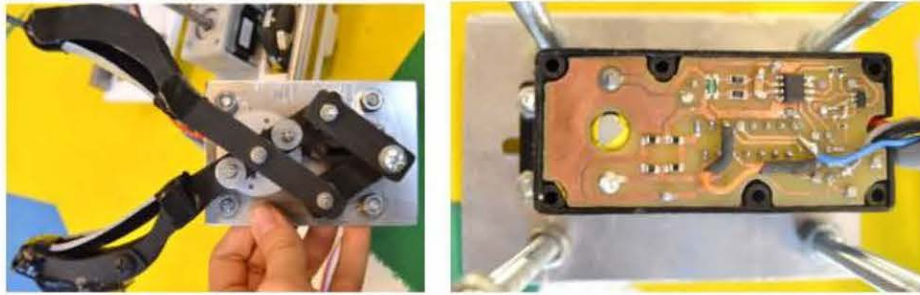


Figure 3. On the left, we can observe the gripper and its rhomboid mechanism. On the right, the new servo motor circuit inserted in its case.

2.3. THE COLOR SENSOR

Finally, a color-sensing subsystem was created. It could not be developed before the competition, but this work was undertaken afterwards because of its pedagogical interest. The main component of this system is a TCS3200, a light-to-frequency converter. It allows to analyze light with four types of photodiodes (red-filtered, blue-filtered, green-filtered and without filter) and gives, as an output, a square wave signal of frequency proportional to the sensed irradiance (Power/Surface). Our goal was to analyze the light reflected by the objects of the playing field and to get information about the red, green and blue content of this light. This information would then be treated in order to distinguish four different colors: white, green, purple (seashells) and orange (wood blocks).

Initial tests made us realize that the sensor was highly perturbed by ambient light. Therefore, we decided to conceive a prototype of the system in which the sensor would be isolated and the observed objects would be illuminated with white LEDs. Other considerations were taken into account, like the fact that the LED light could not directly illuminate the sensor. The resulting prototype consisted on a PCB and a plastic case mounted on it, that can be observed on Figure 4.

The prototype was tested and worked as expected. From these tests we also concluded that, since not all frequencies are emitted with the same intensity by white LEDs, the different outputs of the TCS3200 needed to be normalized.

The TCS3200 output frequency was initially measured with an oscilloscope. Later, we created a program to measure it with an Atmega328p (using the Arduino function *pulseIn()*). After this, we conceived a program to identify colors. It is based on a vector system. Each color is represented by a vector of three components (RED; BLUE; GREEN) and several known colors are stored in memory. When a new color is measured, its vector is compared to each vector in memory (by computing Euclidian distances) and it is identified with the color corresponding to the nearest vector. The program was tested with the playing field elements and could identify all colors without any problems.

The last thing to do was to integrate the color-sensing subsystem in the gripper. To this end, the slave card code was modified to include the new functionality and a new mechanical claw was manufactured.

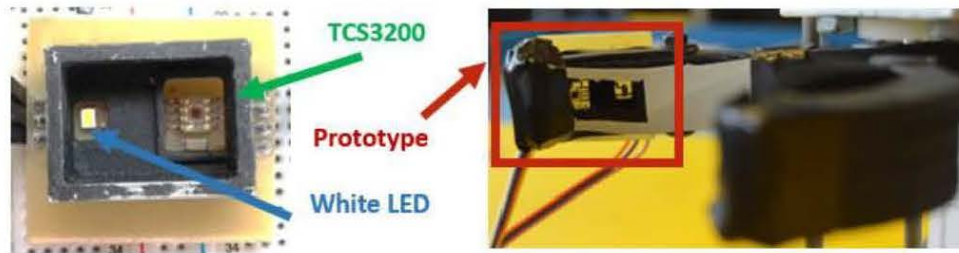


Figure 4. On the left, the color-sensing system prototype. On the right, the same prototype, installed on the gripper.

3. RESULTS

We were able to accomplish a functional system before the contest. The elevating and gripping subsystems worked as expected. Their interaction with the master card was also tested and gave satisfactory results. However, the lack of time made us not identify some problems soon enough; they were detected during the contest.

Firstly, when the stepper motor was powered but not used, it was consuming a large amount of current, since H-bridges were constantly feeding the motor coils. This was improved because, thanks to the screw-nut union, the elevator could be sustained without powering the stepper motor.

Furthermore, the gripping subsystem always work as expected. Only the mechanical system got eventually blocked, what we solved by greasing the gripper.

With respect to the color-sensing subsystem, it provided highly satisfactory results. Even when installed on the gripper, where ambient light could reach the sensor (depending on how the gripper grabbed its load), it was always able to detect the right color.

4. CONCLUSIONS

The conception of this module has been an especially comprehensive job, which has allowed us to develop numerous skills in electronics and other areas of knowledge. With this project, we have been able to build a complete system and to work in every phase of its conception process. It is necessary to highlight the importance of the integration of mechanics, electronics and programming: a fairly difficult task present in any actual industrial project. We have also gained the opportunity to accomplish non-technical but very useful tasks and of great interest (such as fundraising).

5. REFERENCES

- [1] Planète Sciences, “Eurobot. The Beach Bots-2016. Rules”, September 2015.
http://www.eurobot.org/attachments/article/51/E2016_Rules_EN.pdf
- [2] Noguet, “GUIDER EN TRANSLATION: la liaison glissière”, April 2009
<http://sitemecanique.free.fr/Documents%20TGE1/Etude%20des%20constructions/Technologie/CI6%20Analyser%20et%20concevoir%20la%20solution%20constructive%20d'une%20liaison/Liaison%20a%201%20degre%20de%20liberte/Cours%20Guidage%20en%20translation.pdf>
- [3] Pololu Robotics and Electronics, Stepper Motor with 18cm Lead Screw
<https://www.pololu.com/product/2689>
- [4] Pololu Robotics and Electronics, DRV8825 Stepper Motor Driver Carrier
<https://www.pololu.com/product/2133>

This report includes the following documents:

DOCUMENT 1, PROJECT REPORT

Part 1: Project report	pages 25 to 83	59 pages
Part 2: Annexes	pages 85 to 133	49 pages

DOCUMENT 2, SUMMARY OF CHARGES

Introduction	pages 137 to 137	1 pages
Quantities	pages 139 to 141	3 pages
Unit prices	pages 143 to 144	2 pages
Partial amounts	pages 145 to 146	2 pages
Total costs	pages 147 to 147	1 pages

DOCUMENT 1

PROJECT REPORT

LIST OF CONTENTS

PART I: PROJECT REPORT

CHAPTER 1: INTRODUCTION	27
1.1. CONTEXT	27
1.1.1. THE CONTEST	27
1.1.2. FIRST SEMESTER WORK	28
1.1.3. SECOND SEMESTER WORK	29
1.2. INITIAL REFLECTIONS ON THE PROJECT	29
1.2.1. ACCURATE DESCRIPTION OF THE PROBLEM	29
1.2.2. SYSTEM SPECIFICATIONS	30
1.2.3. REVISION OF THE EXISTING SOLUTIONS TO THE PROBLEM AND FIRST CHOICES	31
1.3. MOTIVATION	33
1.4. OBJECTIVES	34
1.5. WORKING METHODOLOGY	35
1.6. RESOURCES	35
CHAPTER 2: THE ROBOTS ARCHITECTURE	37
2.1. INTRODUCTION	37
2.2. THE MASTER CARD	38
2.3. THE GENERAL POWER-SUPPLY CARD	38
2.4. THE BUS	38
2.5. THE SLAVE CARDS	39
CHAPTER 3: THE ELEVATING SUBSYSTEM	41
3.1. ELEVATOR MECHANICS	41
3.1.1. A MECHANICAL PROBLEM: OVER-CENTER LOCKING	42
3.1.2. INSTALLATION OF THE ELEVATOR IN THE ROBOT	43
3.2. ELEVATOR MOTORIZATION	44
3.2.1. STEPPER MOTOR CHOICE	44
3.2.2. FIRST OPTION: THE LINEAR ACTUATOR 42DB10LC2U	44
3.2.3. SECOND OPTION: THE LINEAR ACTUATOR SYS42STH38-1684A	46
3.2.4. STEPPER MOTOR CONTROL	47
3.3. ELEVATOR SLAVE CARD (STEPPER MOTOR CONTROL AND POWERING CIRCUIT)	49
3.4. ELEVATOR CODE	49
3.5. CONTEST RESULTS AND IMPROVEMENTS	49
CHAPTER 4: THE GRIPPING SUBSYSTEM	51
4.1. GRIPPER MECHANICS	51

4.1.1. FINGERS DESIGN	52
4.2. GRIPPER MOTORIZATION	53
4.2.1. SERVO MOTOR CHOICE.....	53
4.2.2. REDESIGN OF THE SERVO MOTOR CONTROL CIRCUIT	54
4.2.3. RETRO-ENGINEERING	55
4.2.4. NEW CIRCUIT.....	57
4.2.5. BREADBOARD CIRCUIT AND FIRST TEST PHASE	61
4.2.6. REDUCED PRINTED CIRCUIT AND SECOND TEST PHASE.....	64
4.2.7. FINAL PRINTED CIRCUIT	64
4.3. GRIPPER SLAVE CARD (SERVO MOTOR CONTROL AND POWERING CIRCUIT).....	65
4.4. LAST TEST PHASE.....	66
4.5. GRIPPER CODE	67
4.6. CONTEST RESULTS AND IMPROVEMENTS.....	67
CHAPTER 5: THE COLOR-SENSING SUBSYSTEM	69
5.1. INTRODUCTION.....	69
5.2. THE TCS3200	69
5.3. DESCRIPTION OF THE PROBLEM.....	70
5.4. FIRST TEST PHASE	70
5.5. PROTOTYPE CONCEPTION.....	70
5.6. SECOND TEST PHASE (PROTOTYPE).....	72
5.7. CODE TO MEASURE THE FREQUENCY	74
5.8. CODE TO DISCERN COLORS.....	75
5.9. INTEGRATION IN THE GRIPPER	77
5.9.1. MECHANICAL INTEGRATION.....	77
5.9.2. CODE INTEGRATION.....	78
5.10. RESULTS.....	78
CONCLUSIONS	79
REFERENCES	81
PART II: ANNEXES	
ANNEX 1: PRINTED CIRCUIT BOARD SCHEMATICS.....	87
1.1. FIRST SHIELD FOR DRV8825.....	88
1.2. ELEVATOR SLAVE CARD.....	89
1.3. SERVO MOTOR CARD	90
1.4. SERVO MOTOR REDUCED CIRCUIT CARD.....	91
1.5. SERVO MOTOR SLAVE CARD.....	92
1.6. COLOR -SENSING SYSTEM PROTOTYPE CARD	93

ANNEX 2: CODE	95
2.1. SLAVE ELEVATOR	96
2.2. SLAVE GRIPPER.....	101
2.3. TEST CLASS COLOR.....	116
2.4. SLAVE GRIPPER EXTENDED (COLOR SENSOR)	120

LIST OF FIGURES

Figure 1. The Eurobot 23 rd edition playing field, based on the theme “The Beach Bots” [1]-	27
Figure 2. Seashell (dimensions in mm) [1]	30
Figure 3. Seashells distribution and field “rocks” (dimensions in mm) [1]	30
Figure 4. The team’s main robot (on the left, containing the seashell-collecting module) and the secondary robot (on the right).	37
Figure 5. Elevator structure.....	41
Figure 6. Elevator mounted on the main robot and loaded with the gripping subsystem and the umbrella.	42
Figure 7. Schematic showing the over-center lock phenomena.....	43
Figure 8. Robot chassis, initial and final versions.	43
Figure 9. Force-speed characteristic of 42DBL10 motors. [17].	45
Figure 10. Control circuit for the stepper motor. Courtesy of https://www.arduino.cc/en/Reference/StepperUnipolarCircuit	46
Figure 11. Pull-out curve of the SYS42STH38-1684A. [19].	47
Figure 12. Schematic of the purchased DRV8825 shield. [21].	48
Figure 13. Initial design of the gripping mechanism.....	51
Figure 14. Rhomboid mechanism of the gripper.	51
Figure 15. The four versions of the gripper fingers (in order of creation, from left to right).	52
Figure 16. Electrical modelling of a DC motor. Courtesy of Polycopié de Systèmes Embarqués, École Centrale Paris (Laurent Cabaret, Philippe Bénabès).	55
Figure 17. Schematic of the servo motor Hitec-805BB original control circuit. .	56
Figure 18. Main elements on the servo motor new circuit.	57
Figure 19. Dimensioning a decoupling capacitor for a servo motor commanded with an H-bridge, model of the problem	58
Figure 20. Current required from the battery by the H-bridge.....	59
Figure 21. Example of a hashing period.....	60

Figure 22. Servo motor low-pass filter test, oscilloscope display.	62
Figure 23. Effect of the parasitic capacitances on the measure of the low-pass filter output voltage.....	63
Figure 24. Instrumental amplifier output voltage.	64
Figure 25. Routing of the complete new servo motor circuit.	65
Figure 26. Prototype of the color-sensing system.	71
Figure 27. Color-sensing system prototype case.....	72
Figure 28. Emission spectrum of the white LED multcomp 703-1026 [30].	73
Figure 29. TCS3200 photodiodes spectral responsivity [7].....	76
Figure 30. Mechanical integration of the color-sensing system in the gripper.....	77

LIST OF TABLES

Table 1. Color-sensing system prototype first tests results: TCS3200 output frequencies (All frequencies in kHz, full-scale frequency : $S_0=1$, $S_1=1$).....	72
Table 2. Normalized results of the color-sensing system prototype first tests...	74
Table 3. TCS3200 output frequencies with configuration $S_0=0$, $S_1=1$: $f_{SAT}=12$ kHz (All frequencies in kHz).	75

PART I

PROJECT REPORT

CHAPTER 1: INTRODUCTION

1.1. CONTEXT

The CRoC (Centrale Robotics Club) is an educational association with a clear annual objective: to build one or several robots that will allow the club to participate in the Eurobot, an international student robotics contest created in 1998. This year, the competition took place from the 5th to the 7th May, at the exhibitions center Les Oudairies (La-Roche-sur-Yon, France).

Therefore, yearly, a group of students of the engineering school CentraleSupélec joins the CRoC, and they turn their participation at the Eurobot into their *Projet Innovation*: a 15-credit second-year subject (equivalent to the 4th year in a Spanish engineering school). Three professors – Nicolas Boullis, Hanane Meliani and Didier Coudray- guide the project. It takes place at the *LISA*, the Informatics and Advanced System Laboratory of the school.

1.1.1. THE CONTEST

The competition is based on two-team matches, during which the robots of both groups perform different tasks in the same playing area: a 3mX2m table provided with several elements (construction blocks, doors etc.) that the robots will use to perform their missions.

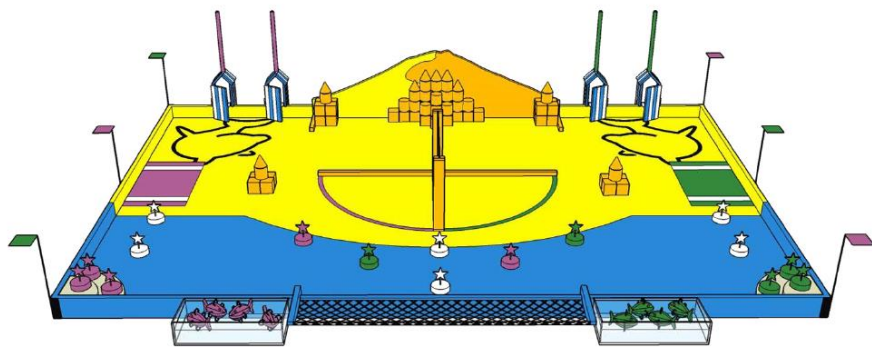


Figure 1. The Eurobot 23rd edition playing field, based on the theme “The Beach Bots” [1]-

In Figure 1 we can see an image of the 2016 edition playing field. In it, we find two water tanks filled with floating plastic fish, wood pieces (cubes, cylinders and cones) that simulate blocks of sand, cabins and seashells represented by hockey disks with decorations on the top. We distinguish two colors (green and purple) indicating to which team each object belongs. Objects in white are neutral (they belong to both teams).

Eurobot rules propose different tasks that the robots can perform during the 95-second matches (the last 5 seconds, during which the robots are not allowed to move, are reserved for them to achieve a “Funny Action”). None of these tasks are obligatory, they are scored differently and can be performed in any order. Regarding our strategy, our team decided to complete:

- Cabin Closing: to close the doors of the team's two cabins.
- Castle Building: to move wood blocks into the central zone of the playing field and, potentially, build a structure.
- Fishing: to move the team's fishes into the net, without transporting any water.
- Seashell Collecting: to transport the maximal number seashells to the team's starting area.
- Funny Action: to open an umbrella installed on the robot.

The rules establish several other limitations which considerably determine the conception of the robot. To have further information on these rules, please consult the contest regulation document [1].

1.1.2. FIRST SEMESTER WORK

During the first semester, thirteen students, including myself, enrolled the project. Our objective for this term was to build a mobile base, autonomous and capable of following a predefined trajectory. To accomplish all the technical work, we started by dividing the team into three groups or sub-teams:

- Captors: to detect and avoid opponents (mandatory) and other objects, this group developed an obstacle-detection system based on ultrasonic sensors.
- Motors: another group was in charge of creating a locomotion system using two DC motors.
- Chassis: a sub-team in charge of designing the mechanical support of the robot.

As the project evolved, new sub-teams were created. In particular:

- Programming: in charge of conceiving the robot's general architecture.
- Powering: a group that designed the circuits needed, on one side, to power the different electronic cards existing on the system and, on the other side, to protect them from powering failures (e.g. an excess of current due to a short-circuit or reverse connections).

Nonetheless, besides the technical work, other tasks had to be performed. More precisely, as the school does not fund the project, it was necessary to find sponsors for the team. We were able to establish new agreements with the team's traditional sponsors (as Schlumberger, Faulhaber or *la Fondation École Centrale Paris*) and to find new sponsors (Mouser and Texas Instruments). In all, the equivalent of 5000 € was collected (in monetary and material contributions). Moreover, we built our own playing field.

Finally, the first term was also the opportunity to learn how to use the softwares Altium, Kicad (PCB design) and SpaceClaim (3D CAD), certain mechanical tools of the laboratory and to learn how to manufacture printed circuit boards.

1.1.3. SECOND SEMESTER WORK

The aim of the second semester was, on the one side, to complete the systems that had been created so far and, on the other side, to build the different modules that would allow the robot to complete tasks during the match. As the project progressed, we realized that we would not be able to integrate all the modules in a single robot (due to the maximum dimensions imposed by the contest regulation). Therefore, it was decided that we would build two robots: a main robot (*Odile*) to perform the “Seashell Collecting”, the “Cabin Closing”, the “Castle Building” and the “Funny Action”; and a secondary robot (*Mitaine*) to perform the “Fishing”.

Regarding my particular situation (a double-degree student having to present the same project in two different universities), it was decided that, during the second semester, I would be in charge of one of the tasks: the “Seashell Collecting”. More specifically, I would be at the head of the group charged with the creation of the seashell-collecting module. The sub-team was composed by myself and two other students, one of which was focused on the system mechanics.

Thus, this end-of-degree project deals with the conception of this particular module of the robot, a task has allowed me to actively participate in all the stages of the process of creation of a complex system.

1.2. INITIAL REFLECTIONS ON THE PROJECT

1.2.1. ACCURATE DESCRIPTION OF THE PROBLEM

Our objective was to conceive a module that would allow the robot to collect the seashells distributed throughout the playing field. In practice, the robot had to transport a maximum number of seashells (neutral seashells or the ones belonging to the team) to the area of departure (the team’s towel).

We decided that this task would be the last one performed, after the “Fishing” and until the robot stopped to open its umbrella.

Each seashell is composed of a hockey disk with decorations on the top. Even though we knew the dimensions of the disk (in Figure 2), it was specified that the form of decorations may not correspond to the drawings presented in the regulation. The only information available was that the ornaments would not exceed a determined cylindrical volume (76.2 mm of diameter and 80 mm of height). The weight of each seashell would never exceed 250 g.

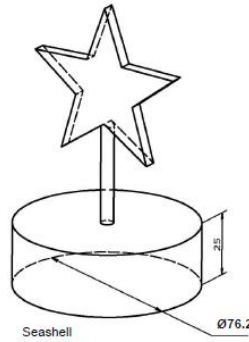


Figure 2. Seashell (dimensions in mm) [1]

There are five possible dispositions of the seashells on the field. In Figure 3, one of the possible configurations is shown. In all five configurations, all seashells are spread over the field's floor except for 6 of them that are placed on the “rocks” (platforms elevated from the floor). The rocks, marked in red in the left image, are shown in detail in the image on the right.

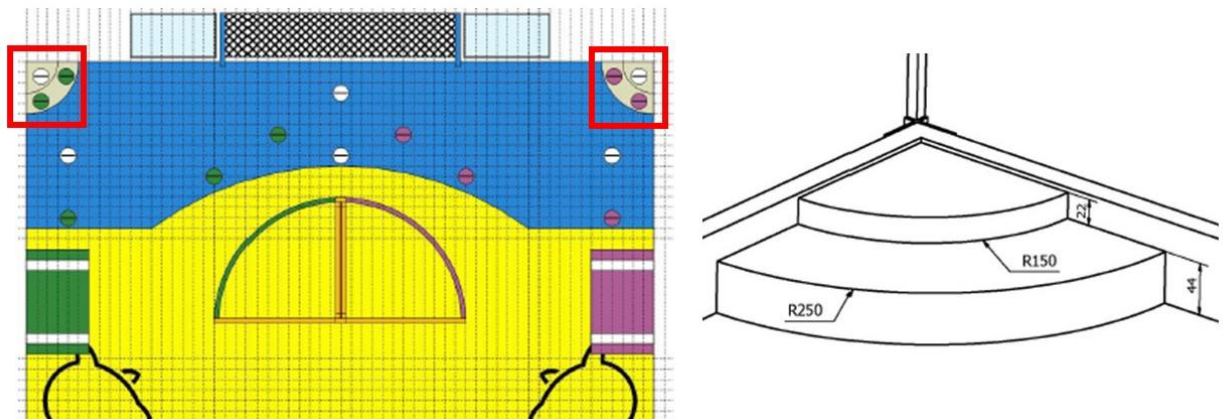


Figure 3. Seashells distribution and field “rocks” (dimensions in mm) [1]

1.2.2. SYSTEM SPECIFICATIONS

To solve this problem, we decided to conceive a module integrating a gripper mechanism. We imposed the following specifications for the system:

- F1. It has to be able to grab and transport the seashells to the team's area of departure.
- F2. It has to detect if the gripper does not catch the seashell properly (and, therefore, if it falls).
- F3. It can only transport the team's seashells or neutral ones.
- F4. It has to be able to move vertically to reach the seashells placed on the elevated platforms.

Besides this, we decided that the module had to fulfill one last function. For the “Funny Action”, another sub-team had developed an umbrella that could be easily opened by pushing vertically its shaft (thanks to a pulley system).

Knowing that the module would require a lift in order to fulfill F4, a last specification was defined:

- F5. At the end of the first 90 seconds of a match, the module's lift has to push the umbrella's shaft in order to deploy it. The shaft can not be moved before this moment.

Finally, we considered the possibility of using the gripper mechanism to transport the sand blocks into the center of the match zone. Nevertheless, we finally opted for using the backside bumper of the robot to push some of the blocks. The gripper would be used if we had the time to develop this possibility before the contest.

1.2.3. REVISION OF THE EXISTING SOLUTIONS TO THE PROBLEM AND FIRST CHOICES

When we started working on this module, the first thing to do was to choose the technical solutions that we would develop to implement the functionalities defined in the previous subsection. Therefore, we devoted some of the first working sessions to getting informed (through web searches and the advice of our teachers) of the solutions that we could implement and to choosing the ones that better suited our needs.

The gripping subsystem (F1)

The gripping mechanism would be entirely conceived and manufactured by us: apart from its suitability to solve our problem, we had to assure the simplicity of the chosen technology.

Firstly, there are several gripping technologies based on attraction forces (such as universal grippers – based on the use of deformable materials and the creation of vacuum - or suction grippers [2], [3]). Besides the fact that it may have been impossible for us to learn how to generate and control this forces before the competition, the design of the gripper head could turn out to be really difficult for us (since these grippers would normally grab the seashells from the top, where an ornament of unknown shape is placed)

Secondly, other systems use needles that penetrate the objects to be lifted. This idea was quickly dropped since it risked of altering the seashells and, therefore, of preventing other teams from performing the task properly.

Finally, the last category studied were the mechanical finger grippers: clamps capable of grabbing and lifting objects thanks to contact forces.

Once again, the lack of information about the ornaments conditioned our choice: we decided to limit our research to two-finger clamps, with which we would be able of encircling the disk without touching its upper side.

Among the possible systems that can action a clamp, we chose to use servomotors, which are perfectly adapted to this project. On the one hand, they can be easily controlled with a microcontroller and perform rotations with great

accuracy. On the other hand, we had found several motors at the laboratory that worked properly and, in any case, we could afford the models that interest us. One last advantage is their compactness.

The elevating subsystem (F4,F5)

In order to elevate the gripper, we needed to conceive a lifting system to transform a rotation, coming from a motor, into the vertical translation of the gripper and all the mechanisms associated. Many different solutions could be implemented to resolve this problem (e.g. a rack-and-pinion system, a pulley mechanism or a screw-nut system).

Our first choice was the screw-nut. This system presents several drawbacks (e.g., big frictions can appear between the threaded rod and the hole). However, good reasons made it the most appropriate for our application. For example, if we compare it to many rack-and-pinion transmissions, the screw-nut assembly lets us move a platform without translating the rods (in our case this could cause a problem, if the rods interfered with other components of the robot due to their movement).

Regarding the motorization, our choice was limited to brushed DC motors and stepper motors. These two options would allow us to obtain continuous rotations and are offered by vendors in France at affordable prices and small sizes.

In this case, stepper motors presented two main advantages: the first one is that we needed no extra components to know the positioning of the load (they can be controlled in open-loop, by counting its steps); the second one, that they had a greater pedagogical interest than the brushed DC motors (since this technology had already been used for the robot locomotion). The drawback was that, with stepper motors, we risked not reaching sufficient speeds.

We decided to start by working on a first prototype using a screw-nut system and a stepper motor. This prototype would later let us verify the viability of the chosen technologies.

Gripping verification (F2)

One of the risks of using a mechanical clamp for the gripping system is that, if the hockey disk is not well placed inside the clamp, the latter can let it fall immediately or some seconds after raising it. It was then necessary to conceive a system that could detect, at any time, if the gripper was grabbing an object. This could be achieved by measuring the pressure exerted by the clamp.

We started by studying the different force sensing technologies existing nowadays [4]. Later, after having consulted the product catalogs of various providers, we concluded that the only affordable options for us were the sensors type FSR [5] (sold, for example, from 7.50 € at gotronic.fr) or the micro-load cells [6] (from 7.90€ by the same vendor). Between these two, FSR seemed the best choice. Their little size and flexibility made them the most appropriate to be installed on the clamp's fingers

Nevertheless, we were encouraged by our teachers to try to exploit the measure of the current demand of the gripper's servo motor, to get information on the torque it applies. This way, we would get information of the presence of an object inside the gripper.

Seashell distinction (F3)

In the Eurobot 23rd edition, before a match started, the referee could arrange the seashells on the field in five different ways. These five configurations were included within the regulation (in every case, the color and the position of each seashell were specified). In order to carry out seashell collection in an efficient manner, we needed to know which configuration was used in each match.

Two possibilities to solve this particular problem were discussed.

- Preconfiguration of the robots. The first option was the installation of a module that would allow to preconfigure the robots in the three minutes of preparation before each match (to choose the seashell configuration and the side of the field the robot would play in).
- Color-sensing system. The second option was to develop a color sensing system that would be installed on the gripper, to detect the color of the seashells. Among the different color sensors existing on the market, we started studying and testing two models available in the laboratory (the TCS3200 [7] and the APDS-9960 [8]).

Undoubtedly, the first possibility was much simpler and faster to implement than the second one, but the latter had once again a greater pedagogical interest. We decided to start by developing a functional module based on the first option and later on, if we had the time, we would develop the second solution.

1.3. MOTIVATION

As well as Eurobot 2016, CentraleSupélec offers to second year students several other projects in which the main objective is to participate in a contest. These are highly valued in the institution because they have a major pedagogical interest. Specifically, the participation in the contest Eurobot yielded enormous benefits for the members of the team. For example:

- We gained the opportunity to design and build a concrete robotic system. This allowed us to truly put into practice many of all theory lessons we have learnt as engineering students and to experience the real practical difficulties of building a robot.
- A robot is a complex system: to achieve this project, we had to develop mechanics, electronics and programming. This way, we have learnt how to integrate these three areas, which coexist in most of actual industrial projects.
- Building a system implies many tasks besides the technical work. Among these, it is important to highlight fundraising: this activity, fundamental for the development of the project, has brought us closer to different companies and allowed us to learn how to deal with sponsors.

Besides all these advantages for the participants, the contest is also beneficial for the industry. In particular, Eurobot sponsors profit from the encounter to get ideas to create new systems and feedback to develop their products.

1.4. OBJECTIVES

We can deduce for the previous paragraphs that this project had one main objective:

- A. To build a functional module that will allow the team to perform the tasks “Seashell Collecting” and “Funny Action” before the Eurobot contest in May 2016.

For this objective to be achieved, several sub-goals needed to be completed. In particular:

- A1. To build the elevating sub-system (create a mechanical design, work on the stepper motor control).
- A2. To build the gripping sub-system (work on the servo motor control, develop the gripping verification system and accomplish a mechanical design for the clamp).
- A3. To build a functional version of the system (integrate the subsystems, test the module and perform needed modifications and adjustments).
- A4. To integrate the system in the robot (install the module in the robot, conceive the seashell collection strategy, implement the master card control, test the integration and perform needed modifications and adjustments).

On the following chapters of this report, we explain in detail how these sub-goals have been accomplished.

Finally, the major objective of the project was completed by two other objectives:

- B. To write the project reports for both universities.

A report, written by all the members of the team and describing all the work done during the year, has been presented to CentraleSupélec. This one will be used as a reference by the future members of the CRoC. In addition, as a personal task, I have written the present report as a part of my end-of-degree project that will be presented to ICAI.

- C. Once having a sure and functional system, to improve the module with new technological approaches.

We have tried to apply these improvements as soon as possible. This has also been a personal task performed during the weeks following the contest.

1.5. WORKING METHODOLOGY

Every week, an entire day of the second-year students' schedule is reserved for projects (Tuesdays). In addition, each student completes its work with overtime as necessary. In particular for the CRoC, the second semester was particularly loaded, especially when the contest date approached. Therefore, it was decided that each one of us would dedicate to the project at least the equivalent to two working days per week (including the reserved time slots).

Reserved time slots were partially used for project planning, since it was the only time the whole team was reunited. Therefore, inside the gripping system sub-team, these hours were used to divide the tasks among the three members, inform about the progression and plan the personal work for the rest of the week. Our teachers were highly available (in person, at the laboratory, or by mail) and we had access to the laboratory every day of the week, including weekends.

In order to manage and share files and code, we use the CentraleSupélec Gitlab platform.

1.6. RESOURCES

The required resources necessary to accomplish the project were:

- The Informatics and Advanced System Laboratory of the engineering school CentraleSupélec (*LISA*).
- Personal computers of the students, in particular the softwares Kicad and Spaceclaim, and the IDE CLion.
- Mechanical and electronical components, donated by the LISA or purchased with the funding from our sponsors.

CHAPTER 2: THE ROBOTS ARCHITECTURE

2.1. INTRODUCTION

Both of the team's robots were built using the same general architecture. Some of its elements have been designed by other sub-teams, different from the one in charge of the module presented with this report. However, it is essential to know the robots organization in order to properly understand the seashell-collecting module description: that is why, within this chapter, we will briefly introduce the robots general architecture.



Figure 4. The team's main robot (on the left, containing the seashell-collecting module) and the secondary robot (on the right).

From an electronic point of view, a master/slave structure was chosen. The robot's electronics have been modularized into different slave cards: each of them is responsible of performing a particular task and includes a microcontroller for doing so. A master card is in charge of controlling all these slave cards and of "organizing" the whole system's activity. The communication between the master and the slaves is achieved thanks to the i2c protocol. A single bus connecting the master and all the slaves simplifies the wiring. All these elements will be described in the following subsections.

With regard to powering, each robot is equipped with a single 12V battery. A general power-supply card transforms these 12V into 5V in order to power the master card, the slave cards and low-power components in general. A brief description of this card is also included in this chapter (subsection 2.3). Concerning high-power elements (i.e. the different motors in the robots), these are powered through specific electronic cards (e.g. to transform the voltage of the battery or to add protection components). Notably, in the main robot, the

powering of the stepper and servo motor of the seashell-collecting module is described in detail in sections 3.3 and 4.3 .

2.2. THE MASTER CARD

Initially, an Arduino board was chosen. Afterwards, this choice changed and the Arduino was substituted by a RaspberryPi [9]. This was mainly because there was a need of a greater capacity for dynamic memory allocation. Also because it was faster and more comfortable to work with a RaspberryPi, since it offers a complete OS we can interact with.

An electronic shield, into which the RaspberryPi is plugged, has been conceived in order to:

- Power the RaspberryPi, with the 5V coming from the general power-supply card. The powering is done through the 5V RaspberryPi GPIO.
- Convert voltage levels. The RaspberryPi works at 3.3V and the slaves (each including an ATmega328P) at 5V. In order for them to communicate (through the i2c or the asynchronous serial protocols), it is necessary to shift 3.3V to 5V, or viceversa, in the wires transporting the signals SCA, SCL etc.
- Connect the master to the bus presented in section 2.4.

2.3. THE GENERAL POWER-SUPPLY CARD

Initially, the card included a linear voltage regulator LM7805 [10]. However, this card needed to be redesigned once the expected current consumption of the system augmented (due to the change of master card and the increment in the number of slaves). Texas Instruments, one of the sponsors of the team, offered us the LM2679 [11]. This is a switch mode regulator that can convert the 12V supplied by the battery into 5V. It offers several advantages with respect to the first model: two examples are its greater efficiency (from 60% to 90% versus 45% to 50% of the LM7805), a greater current supply capacity (up to 5 A versus 1 A of the first model) and that it allows to program a peak current limit.

2.4. THE BUS

As it has been previously explained, by using a bus we would avoid having to connect the master to each slave separately. In addition, this bus allows to transmit other signals (as the 5V and GND powering signals coming from the general power-supply card), which has simplified the wiring even more.

It is a ribbon cable with 14 wires:

- Two wires to transmit signals RX and TX, to establish an asynchronous serial communication between the master and each one of the slaves (for the debugging of the slave's code via the RaspberryPi)

- Two wires to transmit signals SCA and SCL, to establish the i2c communication between the master and all slaves. Thanks to this protocol the master controls the actions of all the robot's modules.
- Two wires to power the cards with 5V.
- Six GND wires to power the cards as well, but also to isolate the different signals in the bus (there is a ground wire between any two other wires).
- Two backup wires, unused.

2.5. THE SLAVE CARDS

Each module has one or several slave cards and each of them performs a different task using a microcontroller. Instead of using already assembled boards, it was decided that we would create the printed circuits from zero. This was mainly due to two reasons: firstly, it would considerably reduce costs; secondly, it allowed to better adapt the dimensions of the cards to our need. We chose to create boards equivalent to the Arduino Uno, since most of the team's members were accustomed to working with the ATmega328P and it has a very affordable price. This is a relative simple task explained by numerous tutorials on the Internet [12].

Besides the specific components of each module, all slave cards include:

- An ATmega328P [13].
- A 16 MHz quartz crystal that acts as an external clock source for the microcontroller (it is completed by two 22 pF capacitors connected from the terminals of the quartz to the ground; the whole is enclosed by a ground plane).
- A no-push button that allows to reset the microcontroller.
- A connector to connect the card to the system's bus.
- Some control LEDs, to verify that the card is powered and that the microcontroller works as expected.
- A 2x3 header, which acts as the microcontroller's ICSP. Through this header, and using an AVR programmer, the ATmega328P can be bootloaded and the programs loaded into it.

CHAPTER 3: THE ELEVATING SUBSYSTEM

3.1. ELEVATOR MECHANICS

Once the technical choices exposed in section 1.2.3 were made, we decided to start by designing and building the elevator's mechanical system. This was a priority task since it conditioned the design of the chassis and the other mechanical elements of the primary robot.

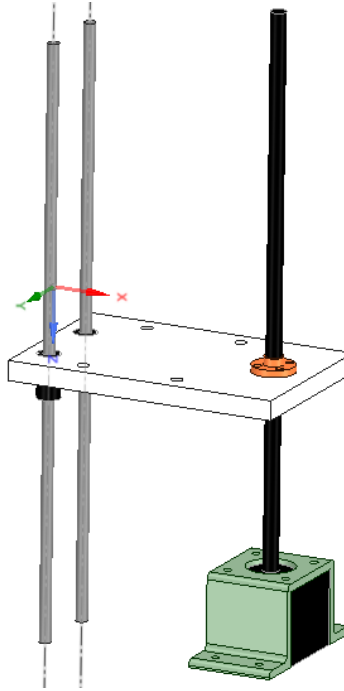


Figure 5. Elevator structure.

The conceived system can be observed in Figure 5. A stepper motor, fixed to the robot's chassis, causes the rotation of a threaded rod (in black) that is, at the same time, inserted in a threaded hole on a platform. Therefore, with this union, the rotation generated by the stepper motor is transformed into the vertical translation of the platform. This movement is also possible thanks to two smooth rods (in gray), which are also fixed to the robot's chassis. The platform slides over these rods as it moves up and down: this reinforces its stability and prevents it from rotating around the threaded rod.

As it has been mentioned before, the function of this elevator is to drag both the gripping subsystem and the umbrella. The lift has been specifically designed to enclose the gripping subsystem under its platform, to which the subsystem is anchored by means of four threaded bolts. Additionally, the umbrella can be placed on the top of the elevator, by attaching its shaft to the platform. The disposition of these two elements can be observed on Figure 6.

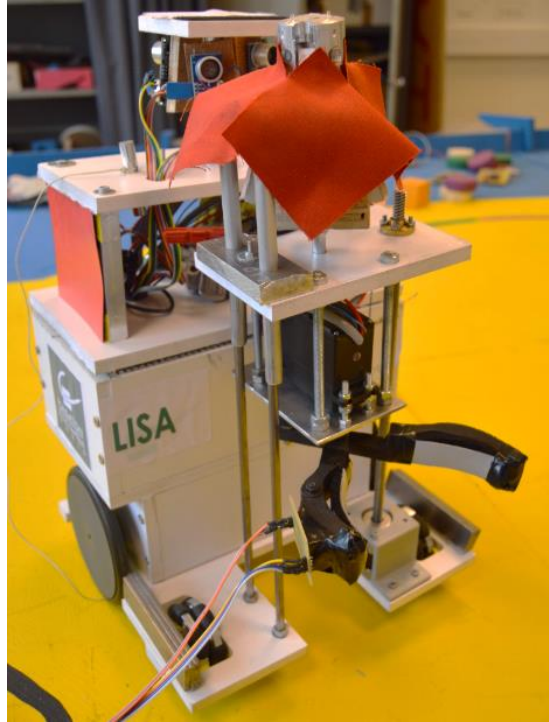


Figure 6. Elevator mounted on the main robot and loaded with the gripping subsystem and the umbrella.

We decided to manufacture the platform out of PVC foam boards. These are the same that have been used to fabricate the robots' chassis and offer significant advantages for the elevator. On the one hand, these are light but resistant sheets: the platform can then drag heavy loads (a servo motor and a hockey disk) without adding extra weight for the stepper motor to lift. On the other hand, PVC is a very machinable material: this allowed to easily fabricate the platform using a CNC milling machine.

Once the stepper motor model was chosen, a support could be manufactured out of PVC. It was later substituted by a final support created with the 3D printer of the CRoC, using PLA.

3.1.1. A MECHANICAL PROBLEM: OVER-CENTER LOCKING

Once the design was accomplished, we manufactured a prototype. This allowed to discover a defect of the system: the platform got blocked due to an over-center lock.

As we have explained, it exists a slide link between the elevator platform and two smooth rods. However, the force that drives the platform (coming from the threaded rod) is applied at a certain distance from the shafts of the smooth rods: due to the gap existing between the platform and these rods, this causes the platform to angle and get blocked (as it is represented on Figure 7).

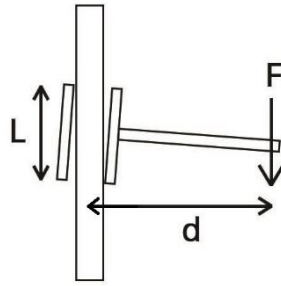


Figure 7. Schematic showing the over-center lock phenomena.

In Figure 7, d corresponds to the distance between the application point of the driving force of the platform (F) and the shaft of the guiding rod. L is the contact length between the rod and the platform.

Initially, we tried to solve this problem installing linear ball bushing bearings between the smooth rods and the platform, but without success.

According to [14], if the gap between the rod and the guided piece is small enough, the over-center lock phenomena will not happen if the following relationship is verified:

$$d \leq \frac{L}{2f}$$

Where f is the static friction coefficient between the rod and the guided piece.

It was then decided to increase the contact length L by inserting two metallic tubes into the platform, shown in Figure 6. Taking aluminum tubes and mild steel rods ($f \approx 0.61$ according to [15]) and knowing that $d=11$ cm, the precedent equation indicates that the length of the tubes must be greater or equal to 13.42 cm.

After implementing this improvement, the platform could be translated by the screw-nut system without problems.

3.1.2. INSTALLATION OF THE ELEVATOR IN THE ROBOT

Even if the robot structure was taken into account in the elevator's mechanical design, the chassis needed to be adapted in order to install this module in the robot. In particular, we passed from an initial chassis shown in Figure 8, on the left, to the chassis on the right.

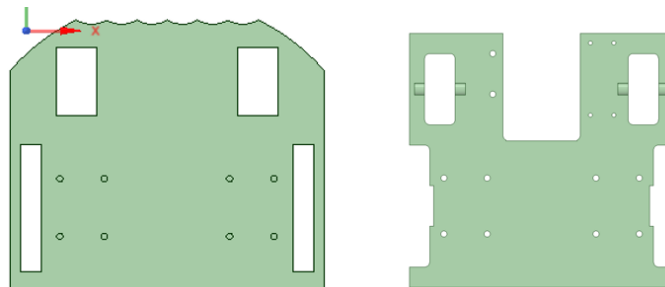


Figure 8. Robot chassis, initial and final versions.

The initial chassis had a bumper: it was supposed to help the robot to push and guide the sand blocs to the center of the field, thanks to the teeth that we can observe on the left. However, tests showed that the bumper did not guide the pieces correctly. That is why it was decided to place the elevator instead and to create a hole that would allow to lower the gripper to the floor level.

3.2. ELEVATOR MOTORIZATION

3.2.1. STEPPER MOTOR CHOICE

The stepper motor choice was undertaken after having accomplished the design of the mechanical system. Three criteria were defined in order to find the appropriate motor for our application:

- Size: the motor would have to be installed in the conceived elevator, without interfering with any other element in the mechanical structure of the robot.
- Speed: the hardest constraint concerning speed was imposed by the “Funny Action”. It was necessary to assure that the stepper motor could raise the platform of 5 cm in less than 5 seconds (in order to completely deploy the umbrella).
- Torque: the stepper motor had to supply a torque capable of lifting the gripping system loaded with a seashell.

The total mass of the elevator and the gripping system was upper-bounded to 1.250 kg and we knew that the mass of the load (a seashell) could not be bigger than 250 g. Therefore, we were able to upper-bound the weight to be lifted to 15 N, that is the linear force that the screw-nut system needed to supply. This data is useful to look for an already assembled linear actuator.

However, it was necessary to estimate the torque required to lift this weight, in case we wanted to look for simple stepper motors (for the purpose of assembling the screw ourselves). We chose then the model proposed by [16], which leads to the following formula to compute the necessary torque (C):

$$C = \frac{F \cdot D \cdot (p + f \cdot \pi \cdot D)}{2 \cdot (\pi \cdot D - f \cdot p)}$$

Where f is the friction coefficient between the screw and the nut, p is the thread pitch, D is the major diameter of the screw and F is the weight of the load to be lifted.

3.2.2. FIRST OPTION: THE LINEAR ACTUATOR 42DB10LC2U

Our teachers offered us a linear actuator used for another project in the past years: a 42DLB10C2U [17]. It is a stepper motor with incorporated screw, so that the assembly can work in two ways: if the motor is fixed, the screw moves linearly; if the screw is fixed and the motor is prevented from rotating, it is the motor that moves linearly. This last option seemed perfect to be implemented in

our system, by fixing the motor to the platform of the elevator. Besides, the nominal voltage of the motor coincided with the voltage of the robot's battery.

It was sure that this motor would be capable of lifting the elevator and its load, since it can lift up to 100 N. However, there were serious doubts whether the speed would be high enough, because of the speed-torque characteristic of the model (Figure 9). The highest speed shown in the graph is of 250 steps/second. Knowing that each step corresponds to a movement of 0.0254 mm, in 5 seconds the platform would move 3.175 cm, not enough to open the umbrella.

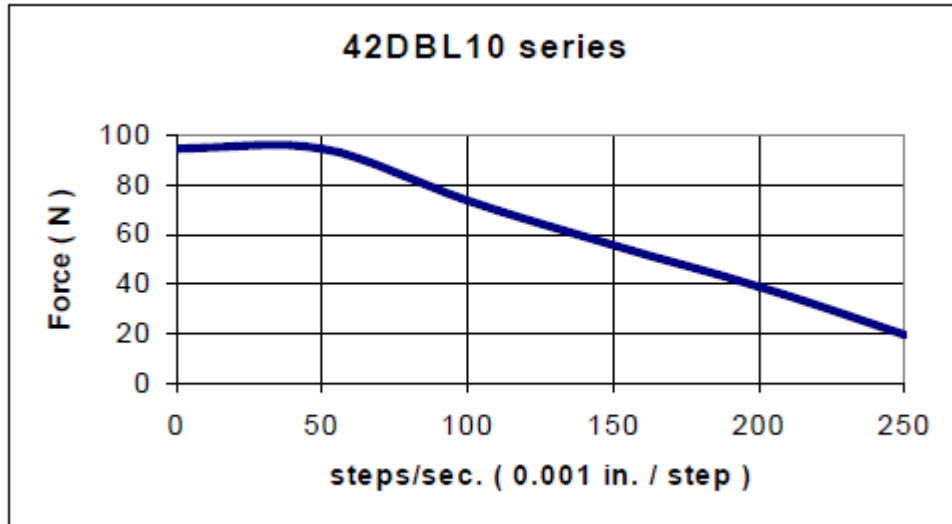


Figure 9. Force-speed characteristic of 42DBL10 motors. [17].

Nevertheless, the instant availability of the motor (it was not necessary to purchase it or to wait for a delivery) pushed us to test it, to see if it was possible to obtain higher speeds and to get a first contact with the driving circuit of a stepper motor.

Since the 42DB10LC2U is a unipolar stepper motor, it can be easily controlled in *wave-step* mode [32]: we just need to let the current pass through each coil, one after another, in a certain order. To let the current pass through the coils, a Darlington array was used (ULN2003A [18] with integrated clump diodes) since this current was expected to be too high to be delivered by an Arduino board.

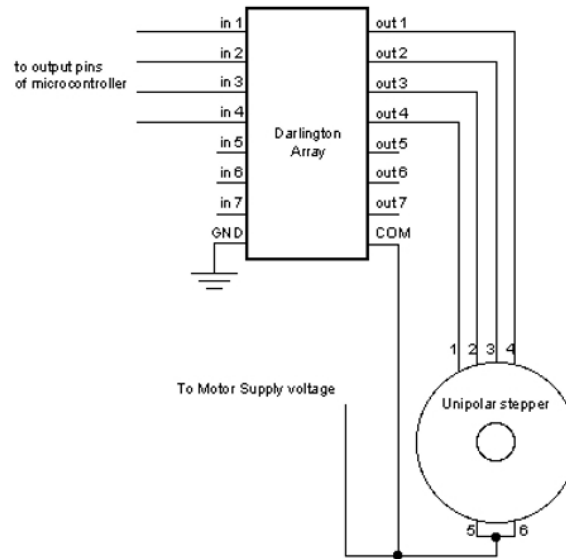


Figure 10. Control circuit for the stepper motor. Courtesy of <https://www.arduino.cc/en/Reference/StepperUnipolarCircuit>

It was confirmed with the tests that 250 steps/sec is the maximal rotation speed of the motor. Beyond this speed, at 300 steps/sec for example, the rotor gets blocked, even if the motor is unloaded. We suppose this is because there is not enough time for a magnetic field to be properly formed in each coil, and then the rotor can not accomplish the steps. Additionally, it is possible that the rotor is “caught up” by a high control speed: a coil is activated but, before the rotor has had the time to make a step, one or more than one other coils are activated too. The rotor is then pushed in different directions and, in result, it vibrates without rotating.

3.2.3. SECOND OPTION: THE LINEAR ACTUATOR SYS42STH38-1684A

After having discarded the first option, we started searching other options in the market. Although both linear actuators based on stepper motors and simple stepper motors were interesting, it was preferable to purchase an already assembled linear actuator. It is important to mention that it was very hard to find one that fulfilled the conditions previously presented (size, speed, torque) and offered by usual suppliers in France (such as Gotronic or Radiospares). Finally, we chose a model proposed by an American manufacturer, Pololu, and offered by Lextronic in France (a SYS42STH38-1684A) [19].

Using the formula presented in subsection 3.2.1 , we can estimate the torque the chosen motor had to supply.

$$C = \frac{F \cdot D \cdot (p + f \cdot \pi \cdot D)}{2 \cdot (\pi \cdot D - f \cdot p)}$$

If:

$F = 15 \text{ N}$ (estimate of the weight of the elevator and its load)

$D = 0.8$ cm (major diameter of the screw [19])

$p = 0.8$ cm (thread pitch of the screw [19])

$f = 0.36$ (kinetic friction coefficient cooper-steel according to [15])

Then the result is 4.6 Ncm.

Knowing that, for this actuator, the linear translation is of 0.04 mm/step, it is also possible to compute the required velocity (using *full-step* control):

$$50 \text{ mm}/5 \text{ sec} \cdot 1 \text{ step}/0.04 \text{ mm} = 250 \text{ step/sec}$$

In this case, it is a bipolar stepper motor with a speed-torque characteristic shown in Figure 11.

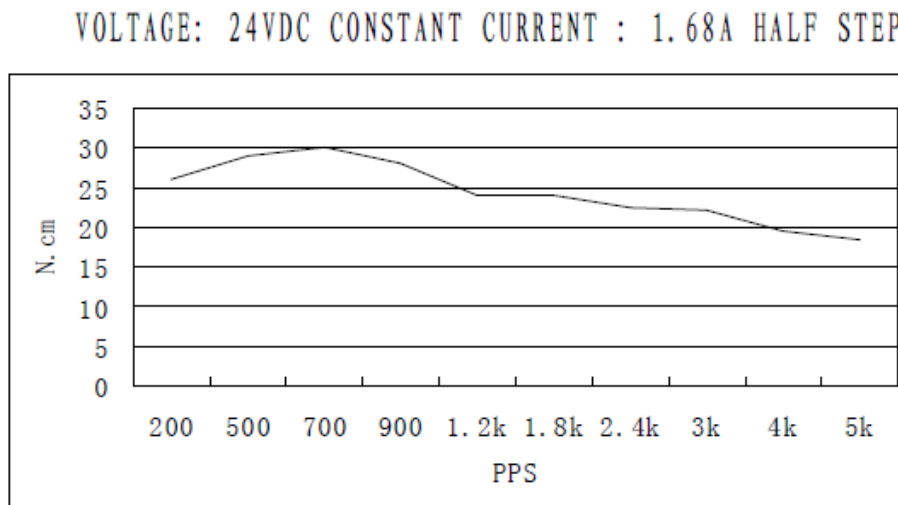


Figure 11. Pull-out curve of the SYS42STH38-1684A. [19].

Looking at the characteristic in Figure 11 we can see that, if the motor is powered with 24 V, it is able to supply a torque of 25 Ncm ($\gg 4.6$ Ncm) at 250 step/sec (the current being limited to the motor rated current, 1.68 A). In our case, the motor would be powered at 12 V. This smaller voltage limits the rate at which the current rises when a coil is activated. However, at small speeds (such as 250 steps/sec) a small voltage does not limit excessively the torque supplied by a stepper motor: at each step, there is enough time for the maximum magnetic field to be established in the activated coil.

Regarding the motor's dimensions, these are significant: however, it was possible to install it in the elevator.

3.2.4. STEPPER MOTOR CONTROL

Being the SYS42STH38-1684A a bipolar motor with two coils, it is not possible to use an array of transistors to control it: two full H-bridges are needed to let the current flow through each coil in two ways.

Instead of looking for a simple H bridge model, we decided to work with a component proposed by Texas Instruments: the DRV8825 [20], a driver for

bipolar stepper motors. It includes two full H-bridges and internal logic circuits to control them. As a result, these bridges can be easily controlled with 4 pins: with MODE0, MODE1 and MODE2 we can specify the stepping mode (*full-step*, *half-step* or different *microstepping* modes [32]) and with STEP we can control when the rotor moves (a rising edge makes the rotor move of one step). In our case, the *full-step* mode has been used: there was no need of precise positioning for the motor load and this mode results in higher linear speeds. The DRV8825 also allows to implement a current regulation, which has been used to limit the maximal current through each coil.

Initially, it was decided to design and manufacture a shield for the DRV8825 at the LISA. In essence, it was a circuit containing the driver, connectors for the inputs and outputs and other components recommended by the datasheet (such as bulk capacitors). We also decided to include a potentiometer connected to pins AVREF and BVREF of the DRV8825. This would let us easily adjust the voltages of these two inputs, which determine the maximum current that can flow through the motor coils (as explained in section 8.3.2 of the component's datasheet [20]). The schematic of this shield is included in Annex 1.1.

However, welding the DRV8825 was a really difficult task that made it impossible for us to manufacture the shield. In fact, this component is only commercialized in HTSSOP28 format: it was too small to be hand-soldered

After three failed PCBs (one of them because of a mistake in the routing, the two others because of the difficulties to sold the component), we decided purchase a card manufactured by Pololu [21]: the schematic, shown in Figure 12 is very similar to the circuit we intended to build.

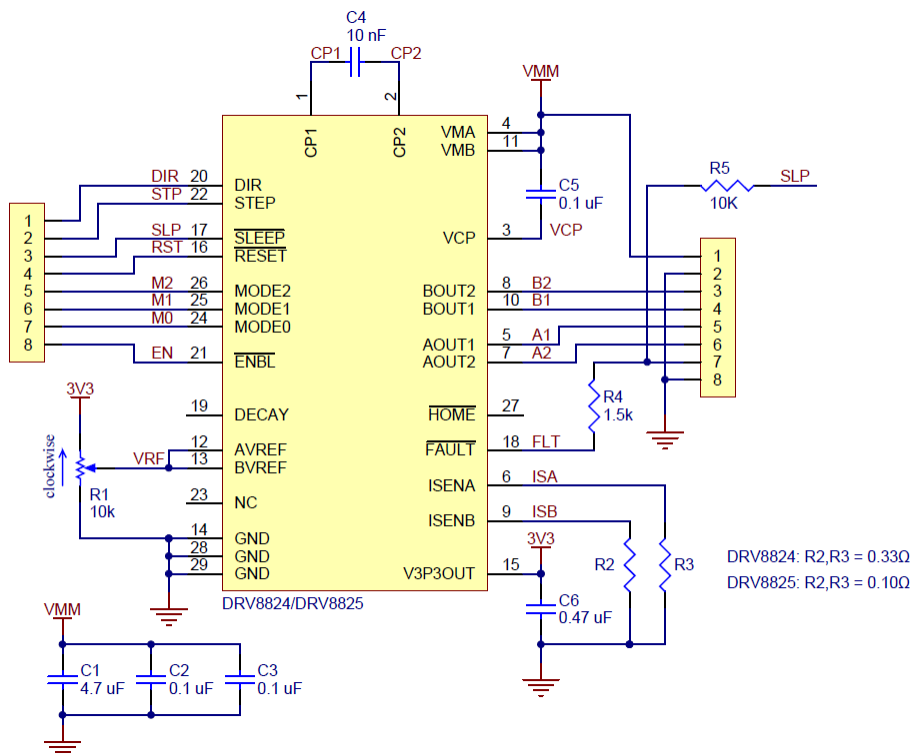


Figure 12. Schematic of the purchased DRV8825 shield. [21].

Even if some components are different, we find almost the same schematic that was initially designed (including the DRV8825, connectors for the inputs and outputs and the resistance and capacitors recommended by the datasheet). It also includes the potentiometer to easily control the maximum current of the motor coils.

3.3. ELEVATOR SLAVE CARD (STEPPER MOTOR CONTROL AND POWERING CIRCUIT)

After having tested the motor and the shield, we designed and manufactured a PCB that enabled the master to control the stepper motor, which is another slave of the system. Its schematic is included in Annex 1.2 . This printed circuit contains the components of any slave card, described in section 2.5 . In addition, it includes female headers that allow to connect the purchased shield to the ATmega328P and to easily install or remove this shield on the slave card (essential to adjust its potentiometer or if it needs to be replaced).

Concerning the motor powering, there was no need to adjust the battery voltage, since it is powered at 12 V. Since the DRV885 already implements overcurrent protection, no extra protection circuitry has been added: the purchased card is directly connected to the battery thanks to a connector on the slave card. This connector includes a mechanical coding system in order to avoid reverse connections.

3.4. ELEVATOR CODE

The code of this subsystem microprocessor is, as in any other slave, object-oriented code. More specifically, we can find a class *Elevator* with which the stepper motor can be controlled by sending different orders. The most characteristic methods of this class are *goTo()* and *update()*. With the first one, we order the position to which the elevator has to move: if it is different from the elevator's position at the moment when the function is called, the function deduces the direction of the required movement and commands it. With *update()*, the height of the elevator is updated and the latter is stopped if it reaches the target height.

The steps are commanded with a PWM signal: the datasheet of the DRV8825 specifies that the stepper motor takes one step for each rising edge on the *STEP* pin. Therefore, f_{PWM} in Hz coincides with the stepper motor speed in step/sec.

These and other aspects are described in detail through comments on the code, included in Annex 2.1.

3.5. CONTEST RESULTS AND IMPROVEMENTS

Even if the system was tested several times before the Eurobot, the lack of time before the contest made us not identify some problems soon enough, and these were detected during the contest.

Firstly, when the stepper motor was powered but not used, it consumed a large amount of current (around 400 mA). This was because the H-bridges were enabled and feeding the motor coils to create magnetic fields that would assure the rotor stayed in the last commanded position. This was unnecessary: even if the motor is not powered, the threaded rod is able to sustain the platform and to prevent it from moving. It is indeed one of the advantages of the screw-nut system: if we apply a torque, it will be easy to transform a rotation into a linear movement but we will not be able to easily transform a linear movement into a rotation by applying a linear force (that is, the weight of the system will not make it drop). We think the great current demanded by the stepper motor, combined with other big current demands from other components in the robot, may have sometimes caused the failure of the robot during the contest (probably, not enough current arrived to the master).

The solution was to use the *(NOT)ENABLE* pin of the motor's shield, in order to disable the H-bridges when it was not used. However, this was done after the contest had finished.

Secondly, the quality of the potentiometer of the purchased shield was unsatisfactory. It was extremely difficult to obtain the desired voltage and it got frequently unsettled (by the robot's movements or its transportation for example). It was later suggested that, once settled, we could add some glue to prevent the wheel from rotating.

We have thought of one possible improvement for the system: position interrupts could be added to the elevator, to create references for its positioning. The system that has been implemented (manually positioning the elevator at the beginning of each match and counting its steps) has caused no problem. However, these switches would provide robustness to the elevator.

CHAPTER 4: THE GRIPPING SUBSYSTEM

4.1. GRIPPER MECHANICS

Once it was decided that a finger gripper would be built, we started by studying the several possibilities we could implement. We considered the option of building a linkage gripper and designed one, shown on Figure 13.

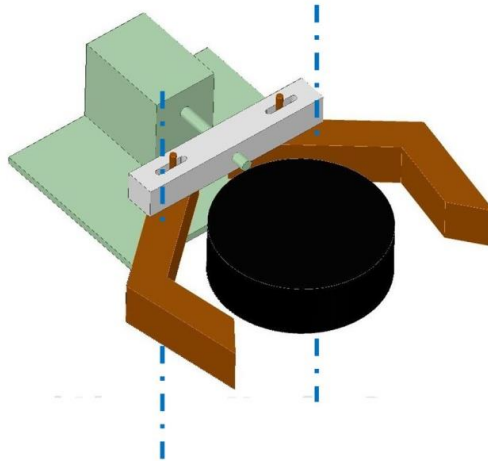


Figure 13. Initial design of the gripping mechanism.

Each of the two fingers turns around a fixed rod (which are not visible from this perspective, their axis are indicated with discontinuous blue lines). In addition, these fingers are joined to a common piece (in gray) by slide links: this way, the forward and the backward movements of the gray piece cause the opening and closing of the clamp.

The advantage of this system is that, in it, a single movement (the linear movement of the gray piece) leads to the symmetric opening and closing of both fingers. The disadvantage is that it requires once again transforming a rotating movement into a linear one. Nevertheless, another solution giving the same result was found: a rhomboid mechanism. This was the solution finally implemented, the result can be observed in Figure 14.

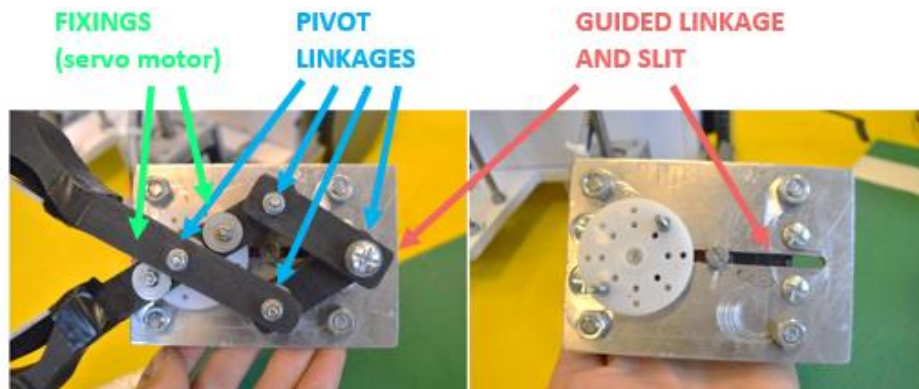


Figure 14. Rhomboid mechanism of the gripper.

In this case, both fingers are joined one to another by a pivot linkage. Additionally, each finger is joined to a rectangular piece and the two rectangular pieces are joined one to another (all the three unions are also pivot linkages). The union point between the two rectangular pieces has a guided movement: the bolt assuring the linkage is introduced in the slit of a plate placed between the servo motor and the gripper. Finally, one of the fingers is fixed to the servo motor rotor, with the union point between the clamp's fingers on the rotor's axis.

With this mechanism, the rotation of the servo motor causes the rotation of one of the fingers. This finger drags the rectangular piece joined to it. By means of the guiding slit, this will generate the symmetrical movement of the other rectangular piece and finger.

In order to validate this second option, we started by creating a prototype of the gripper (using wood fingers and a metallic plate) and testing it. For that, a servo motor Hitec-805BB [33] was used (the model chosen for the system, as explained in the following sections). These tests showed that all linkages worked as expected and that the system was capable of seizing a hockey disk without letting it fall, even if exposed to mechanical shocks. In view of these satisfactory results, we decided to keep the rhomboid mechanism for the final system.

4.1.1. FINGERS DESIGN

Even if the first design of the gripper was proved to be able to grab and hold a hockey disk, we continued developing the form of the fingers.

We can observe the four versions of the gripper that have been developed in Figure 15 (the rhomboid mechanism has always been used).



Figure 15. The four versions of the gripper fingers (in order of creation, from left to right).

On the left, the prototype. In the second model, the fingers were lengthened in order to reach the seashells placed on the rocks. However, with this gripper the robot did not respect the perimeter limits, which is why we were forced to develop the third model (the one that was taken to the contest). In it, the fingers length has been considerably reduced, but their shape has been improved to perfectly match the shape of a hockey disk. This increases the contact surface

between the gripper and the seashell and, therefore, enhances the sustentation. In addition, the contact surface has been covered with a silicone sheet to enlarge the sustentation even more. Finally, the forth model was an improvement developed after the contest. Its shape is very similar to the third model shape, but this gripper includes a color sensor (that will be developed on Chapter 5). In addition, foam rubber has been included between the silicone sheets and the fingers (to enable the system to catch more easily objects other than hockey disks – such as cones or cubes).

4.2. GRIPPER MOTORIZATION

4.2.1. SERVO MOTOR CHOICE

As it is specified on section 1.2.3, it was decided to use a servo motor to operate the gripper. Once again, we started by considering the servo motors that the LISA offered us.

This time, one parameter limited the model choice:

- Torque: it was necessary to choose a motor capable of delivering the torque necessary to grab and raise a hockey disk. We then needed to compute an estimate of this torque.

For this estimate, we can consider that all the force exerted by the gripper fingers is concentrated on a single contact point between the gripper and the hockey disk. This force needs to be big enough to not let the disk slide or fall from the gripper: for that, the frictional force between both objects has to be bigger than the weight of the disk.

This means:

$$P \leq \mu_s \cdot N$$

Where P is the weight of the hockey disk, μ_s is the static friction coefficient between the gripper and the disk and N is the force exerted by the gripper, normal to the disk surface at the point of contact.

Knowing that this force is equal to the torque exerted by the servo motor (C) divided by the distance from the servo motor axis to the point of contact (d), we obtain:

$$C \geq \frac{P \cdot d}{\mu_s}$$

Considering:

- $\mu_s \approx 0.7$ (wood (gripper) on rubber (hockey disk) static friction coefficient – NOTE: this estimate was computed before adding silicone sheets to the gripper).
- $P = 0.250 \cdot 10 = 2.5 \text{ N}$ (upper-bounded weight of a seashell)

- $d = 0.10$ m (upper-bounded distance between the axis of the servo motor and the points of contact of the gripper fingers and the grabbed object)

We obtain a minimum torque of 0.36 Nm.

Two motor models were found at the laboratory: Hitec-805BB [33] and the Futaba-S3003 [34]. According to the specifications of their datasheets, only the first could deliver the required torque (it is supposed to deliver up to 1.94 Nm).

The tests mentioned on section 4.1 served also to verify the precedent estimate. The good results already discussed corroborated that the Hitec-805BB was capable of raising a hockey disk without letting it fall.

4.2.2. REDESIGN OF THE SERVO MOTOR CONTROL CIRCUIT

As anticipated in the preliminary studies, controlling the servo motor position was insufficient. This would have only let us control the opening angle of the gripper without knowing whether it was grabbing an object or not. For that, it was necessary to measure the current flowing through the servo motor, which is proportional to the couple exerted in a DC motor (such as the one found in the Hitec-805BB).

Initially, we thought of conceiving an intensity control and superposing it to the original position control of the servo motor. However, we did not know the controller type implemented in the servo motor (and it can not be known: we could not find any public information on the internal electronics of Hitec servo motors. Even opening them, we were unable to identify some of its components). This could cause problems because of the use we wanted to make of the servo motor, and these could worsen if we tried to superpose another controller.

With the position control, we would command a certain opening angle to make the gripper grab a disk. This angle would have to assure that the motor forced the gripper continuously: it would then be an angle slightly smaller than the one corresponding to the dimensions of a hockey disk. Therefore, the gripper would never reach the commanded position (because the disk would block its closing) and would continuously exert a force on the disk.

If the controller was of type PID, this would cause the “explosion” of the integral term: if the measured position never reached the commanded position, the integral of the difference between these two terms would grow unceasingly.

The control signal for a DC motor is the voltage applied between in its terminals. If a PID was implemented to control the motor, this voltage would correspond to:

$$U(t) = Kp \cdot e(t) + Kd \cdot e'(t) + Ki \cdot \int e(t)dt$$

Where $U(t)$ is the motor's voltage, $e(t)$ is the difference between the measured and the commanded position and K_p , K_d , K_i are the coefficients of the controller. If the integral term grows unceasingly, the tension will do it too.

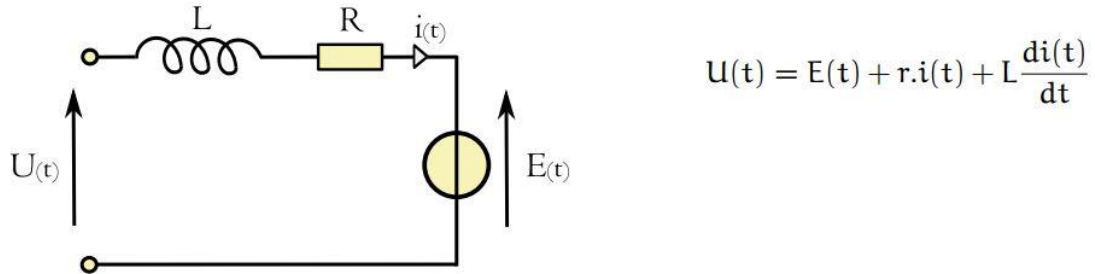


Figure 16. Electrical modelling of a DC motor. Courtesy of *Polycopié de Systèmes Embarqués, École Centrale Paris (Laurent Cabaret, Philippe Bénabès)*.

Taking the equations from the mechanical and electrical models of DC motors (Figure 16), we can deduce:

$$U = E + R \cdot i(t) + L \cdot \frac{di(t)}{dt} = k \cdot \Omega(t) + R \cdot \frac{C(t)}{k} + \left(\frac{L}{k}\right) \cdot \frac{dC(t)}{dt}$$

R corresponds to the total resistance of the rotor and the contact between the brush and the commutator, L is the self-induction coefficient of the rotor, E is the back electromotive force and $i(t)$ is the current flowing through the rotor.

$\Omega(t)$ and $C(t)$ are the rotor rotation speed and exerted torque. If the gripper is blocked, $\Omega(t)=0$ and $U(t) = R \cdot C(t) / k + (L / k) \cdot dC(t) / dt$, with an increasing $U(t)$. Then, the “explosion” of the integral term would cause the “explosion” of the rotor couple and current. This could damage the motor's mechanical and electrical systems.

It was then decided that we would remove the original control circuit of the servo motor and build one ourselves. The first objective was to conceive a circuit that would let us drive the motor, measure the rotation angle of its axis and the current flowing through it. Later on, a numeric control would be implemented using this circuit and a microcontroller.

4.2.3. RETRO-ENGINEERING

The first step was to analyze the original control circuit, in order to determine if some of its components or if its structure could be reused for the new control circuit.

The servo motor had stopped working during one of the tests prior to this phase so, before starting it, we unsoldered and tested the DC motor, which worked

properly. This ensured that the error concerned the PCB and that the motor could be reused in the new assembly.

After that, we tried to deduce the electronic schematic corresponding to the circuit and to identify the components used in it. We obtained the schematic presented in Figure 17.

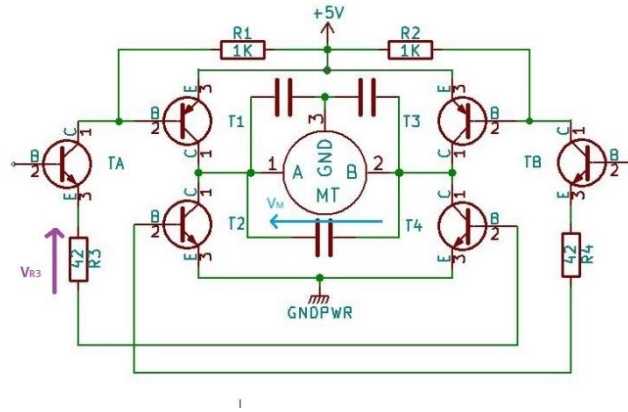


Figure 17. Schematic of the servo motor Hitec-805BB original control circuit.

The components beyond transistors A and B, used to implement a controller for the DC motor, could not be identified. All transistors are BJT: A and B are MMBT2222 [22], 1 and 3 are KSA1010 [23] and 2 and 4 are KSC2334 [24].

This circuit works as follows: if, for example, a voltage is applied to the base of transistor A (bigger than the sum of voltages V_{BE} of transistors A and 4), a positive voltage V_{R3} appears: this generates the base current of transistors 1 and 4 and activates them. This way, if V_{EC} on 1 and V_{CE} on 4 are not too high, $V_M \approx 5V$. If, on the contrary, we apply a zero voltage to the base of A, this transistor is blocked and, therefore, transistors 1 and 4 are blocked too: no voltage is applied to the motor. The functioning is the same for transistors B, 2 and 3: if we apply a sufficient voltage to B, $V_M \approx -5V$; if we apply a zero voltage, no voltage will be applied between the motor terminals.

We were surprised to discover that the circuit did not include free-wheeling diodes, which allow the continuity of the current through a motor when transistors switch.

After this study, we unsoldered the components and build the same circuit on a breadboard, to test it. However, the circuit did not work (which make us think that the error that impeded the servo motor to work is in these components). The near date of the contest made us fast choose an already assembled H-bridge (instead of spending time on searching the cause of the error).

4.2.4. NEW CIRCUIT

We can observe the structure of the new circuit in Figure 18:

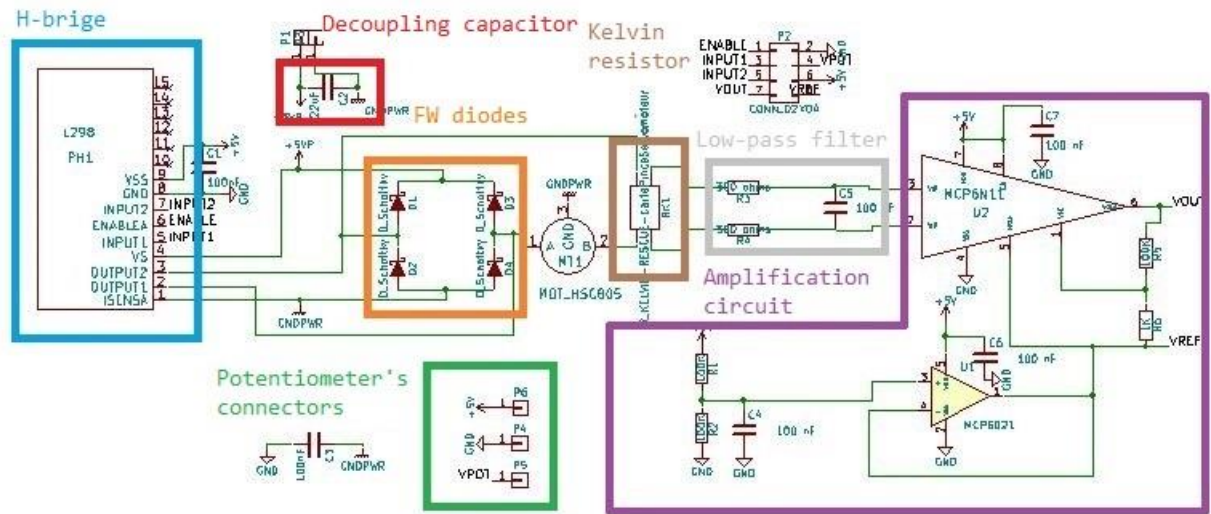


Figure 18. Main elements on the servo motor new circuit.

Another image of this schematic, in greater detail, is included in the Annex 1.3.

The main elements of the circuit are:

- **An H-bridge**, a L298H [25]. It is specified in the manufacturer's documentation that the powering voltage of the component (the one that powers the motor connected to it) should be at least 7.5 V. This raised a problem for the motor that we wanted to use, that has to be powered with a voltage from 4.8 V to 6 V. However, we found some applications in which a L298H is powered with 5V. We verified ourselves that this was possible before finally deciding to use the component.

- **Free-wheeling diodes**. We chose Schottky Diodes (low forward voltage drop).

Initially, for the test circuit (built on a breadboard) we had chosen the model MBR3050CT [26], immediately available at the laboratory, but these were over dimensioned for our needs. That is why the diodes BAT60A [27] were purchased for the final circuit, more suitable for our application.

In fact the MBR3050 allow a maximum forward current of 30 A and a maximum reverse voltage of 50 V. Meanwhile, the BAT60A allow up to 3 A of forward current and 10 V of reverse voltage, enough for the circuit. On the one hand the H-bridge allows up to 2 A in DC operation. On the other hand, later tests showed that the motor's current never went beyond 1.5 A. Regarding the reverse voltage, it is much higher than the saturation voltage V_{CE} of the L298H transistors ($MAX V_{CE}=2.7 V$).

- **A decoupling capacitor, of great capacitance (22 µF)**, to prevent the variations in current generated by the motor to affect the powering of

other elements in the system. In fact, there are inductive effects in the circuit between the battery and the motor. If the current the motor requires varies, the current flowing through these inductances varies too and this generates unwanted voltage drops.

The inductive character of DC motors (that causes the current flowing through it to vary in time) can generate voltage drops. However, in this case, the main responsible for the voltage drops would be the hash of the motor.

During one hash period, there are two phases. During a certain time, the current flowing through the motor comes from the battery and its windings get “charged” (due to the inductive character of the motor). When the transistors commute, this current does no longer come from the battery: it comes from the motor windings, which discharge. We then can consider that there are two “current sources” for the motor. Therefore, the current required from the battery changes drastically and at high frequencies, which can generate significant voltage drops. To avoid it, a decoupling capacitor can be placed between the battery and the motor, which will mitigate the large fluctuations of the current demanded to the battery.

DIMENSIONING THE CAPACITOR:

The problem described above can be modelled with the simplified schematic shown on Figure 19.

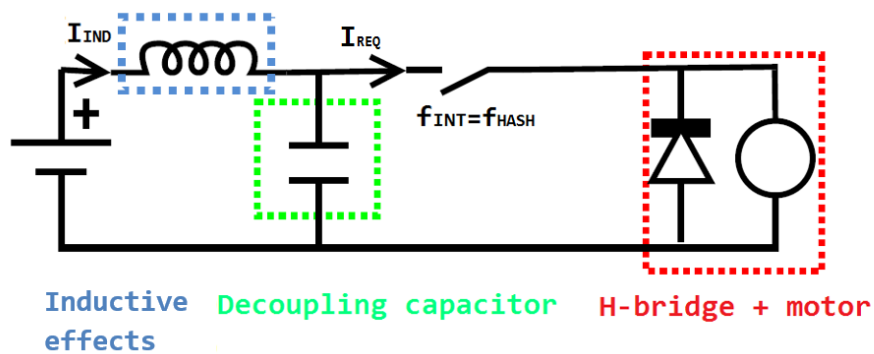


Figure 19. Dimensioning a decoupling capacitor for a servo motor commanded with an H-bridge, model of the problem

The role of the capacitor is to “substitute” the battery, by charging the motor’s windings with current when necessary. Ultimately, all this current comes from the battery but using a capacitor, which acts as an intermediary charge repository, can be very beneficial. In fact, the battery charges the capacitor continuously and this one gets periodically discharged to feed the motor. Therefore, it is the current going from the capacitor to the motor that varies drastically, and not the one coming from the battery. Then, if the capacitor is placed near the motor, the highly variable current will avoid most of the parasitic inductances.

If we simplify and consider, for this estimate, that the current going through the motor is constant and equal to i , the current that the H-bridge requires from the battery is:

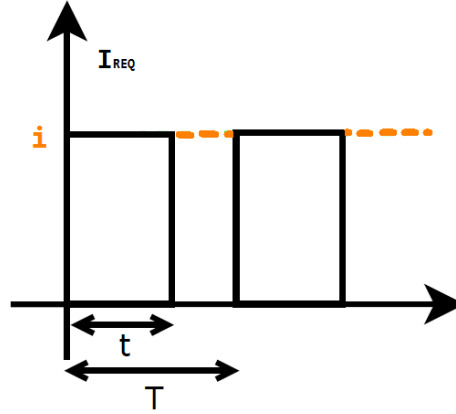


Figure 20. Current required from the battery by the H-bridge.

Where T is the inverse of the hash frequency ($T = 1 / f_{HASH}$) and $t = a \cdot T$, where a is the duty cycle.

If C is the decoupling capacitor capacitance and Q its charge, we have:

$$Q = C \cdot U \leftrightarrow dQ = C \cdot dU \leftrightarrow \Delta Q = C \cdot \Delta U \leftrightarrow$$

$$I_{COND} \cdot \Delta t = C \cdot \Delta U \leftrightarrow (I_{IND} - I_{REQ}) \cdot \Delta t = C \cdot \Delta U$$

During the discharge of the capacitor $I_{REQ} = i$ and $\Delta t = a \cdot T = a / f_{HASH}$. In addition, we can consider that the current charging the capacitor is constant and equal to the average current required by the motor (i.e.: $I_{IND} = i \cdot a$). Then, we obtain that the required capacitance is:

$$(-\Delta U) = \frac{i \cdot \alpha \cdot (1 - \alpha)}{C \cdot f_{HASH}}$$

ΔU (a negative magnitude) corresponds to the drop of the capacitor voltage when it discharges. If we want to assure a maximum drop of Δu^* , then the value of C has to be:

$$(-\Delta U) = \frac{i \cdot \alpha \cdot (1 - \alpha)}{C \cdot f_{HASH}} \leq -\Delta u^*$$

$$C \geq \frac{i \cdot \alpha \cdot (1 - \alpha)}{(-\Delta u^*) \cdot f_{HASH}}$$

We know $f_{HASH} \approx 31,4 \text{ kHz}$ and we can take $\Delta u^* = -0.5 \text{ V}$ (10% of 5V, powering voltage for the motor).

If we want to be sure C is big enough, we can compute the upper-bound of the previous quantity: for that, we can take $i = I_{MAX H BRIDGE} = 2 A$ and $a = \frac{1}{2}$ (because this value maximizes $a \cdot (1 - a)$). With those values:

$$C \geq 32 \mu F$$

However, we had at the laboratory low-ESR $22 \mu F$ capacitors that had been used for to decouple the DC motors used for the robots locomotion. We decided to use these initially. This ensured:

$$\Delta U = \frac{i \cdot a \cdot (1 - a)}{C \cdot f_{HASH}} \leq \frac{I_{MAX H BRIDGE} \cdot 0.5 \cdot 0.5}{22 \cdot 10^{-6} \cdot 31.4 \cdot 10^3} \approx 0.72 V$$

Later, we would have to verify if the decoupling was enough.

- **A kelvin resistor**, to measure the current flowing through the motor by measuring the voltage between its terminals. In opposition to what is recommended by the datasheet of the L298H, we did not connect this resistor between the pin SENSE and the ground, but in series with the motor. Actually, because the motor is hashed, the average current passing through this two points is not the same. This is illustrated by Figure 21.

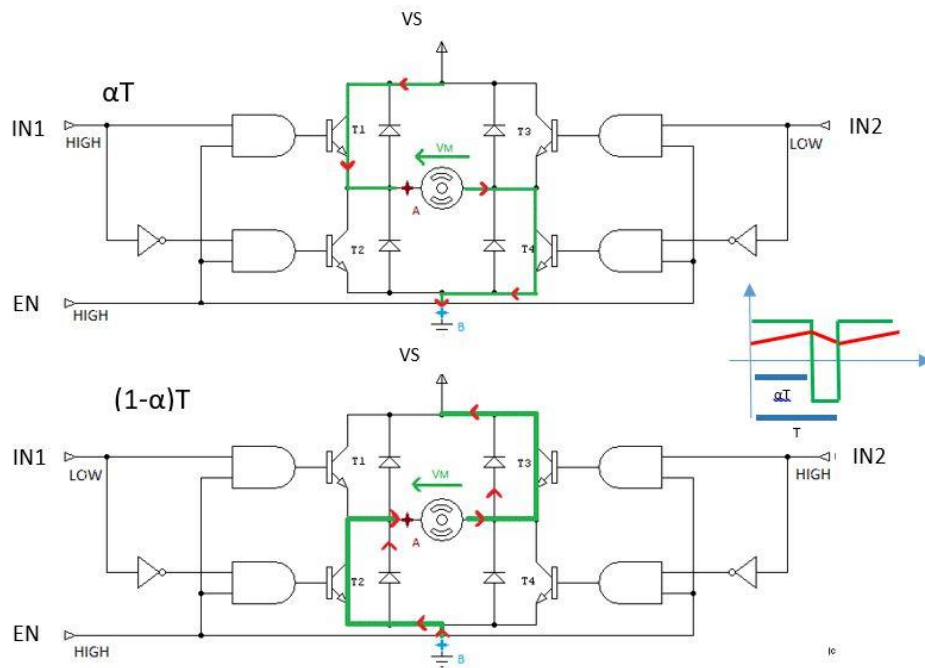


Figure 21. Example of a hashing period.

We can observe that, in a period T , the average current through points A and B is not the same. It is more interesting to place it at point A, since the current passing through this point is exactly the same flowing through the motor.

Kelvin resistors are interesting because of two reasons. Firstly, because their resistance is very low (0.01Ω in this case) and, therefore, barely lessen the voltage applied to the motor.

Secondly, they have four terminals: the intensity that wants to be measured goes through two of them and the other two are used to take the voltage measure (the current going through these last two terminals has then very low values with respect to the first one). All four terminals have parasitic resistances (e.g. in the welds and copper traces) of values non-negligible with respect to the resistor value ($0.01\ \Omega$). However, these will not disturb the measure: since the current going through the parasitic resistances is very low, the voltages they generate are negligible with respect to the voltage we want to measure ($0.01\ \Omega \cdot I_{\text{motor}}$).

- **A low-pass filter.** The PWM control applied to the motor and its inductive character make the current flowing through it oscillate. Therefore, the voltage difference between the terminals of the kelvin resistor does too. However, what we need is a measure of the average intensity going through the motor (a signal as flat as possible, because it will later be treated by a microcontroller: the oscillations and peaks would skew the results). The low-pass filter on Figure 18 was chosen in order to attenuate the hash frequency (approximately 30 kHz). Since the cutoff frequency of this filter is $f_c = 1/(2 \cdot \pi \cdot C_5 \cdot (R_3 + R_4))$, we have chosen $R_3 = R_4 = 270\ \Omega$ and $C_5 = 100\ \text{nF}$ to get $f_c \approx (30/10)\ \text{kHz}$. This filter is symmetric since noises can come from both terminals of the kelvin resistor.
- **An amplification subcircuit,** to amplify the voltage between the terminals of the kelvin resistor. In it, we find an instrumentation amplifier (a MCP6N11 [28]). The amplified voltage corresponds to $V_{\text{OUT}} = \text{GAIN} \cdot V_{\text{IN}} + V_{\text{REF}} = (1 + R_5/R_6) \cdot V_{\text{IN}} + V_{\text{REF}}$. The maximum and the minimum possible value of the current through the motor are, respectively, 2A and -2A (H-bridge limits): the range of V_{IN} is then $[-0.02, 0.02]\ \text{V}$. If we choose $\text{GAIN} = 101$ and $V_{\text{REF}} = 2.5\ \text{V}$, the range of V_{OUT} will be $[0.5, 4.5]\ \text{V}$. This can be correctly processed by an analog pin of an ATmega328P.

4.2.5. BREADBOARD CIRCUIT AND FIRST TEST PHASE

Before creating a PCB, we estimated that it would be convenient to test the previous circuit on a breadboard, in order to verify that the conceived filter was good enough to smooth out the measure. As the instrumental amplifier chosen is only manufactured in CMS format, it was not included and we planned to directly measure filter output voltage (using an oscilloscope: the probe on one terminal and the ground on the other). Nevertheless, this turned out to be a bad decision, the first results were disappointing:

- On the first place, it was impossible to smooth out the filter output voltage.

Our aim with this measure was to be able to determine when the motor is mechanically blocked and can not turn (since, when the gripper grabs an object, the load prevents it from closing). Therefore, our concrete objective in this phase was to obtain a current measure with which we could

distinguish two states: a motor running on empty and a motor mechanically blocked.

We started by testing the filter presented in the previous subsection ($R_3 = R_4 = 270\ \Omega$ and $C_5 = 100\ \text{nF}$ to get $f_c \approx (30/10)\ \text{kHz}$). However, the results were not the expected: the noise amplitude was greater than the difference between the average kelvin resistor voltage with an idle and a mechanically blocked motor.

We decided then to decrease the cutoff frequency of the filter. We increased the values of R_3 , R_4 and C_5 several times and tested the resulting filters, but always with disappointing outcomes. We can observe, on Figure 22 the results obtained with the tested filter with the lowest cutoff frequency: $R_3 = R_4 = 2.7\ \text{K}\Omega$ and $C_5 = 47\ \mu\text{F}$, so $f_c \approx 0.63\ \text{Hz}$.

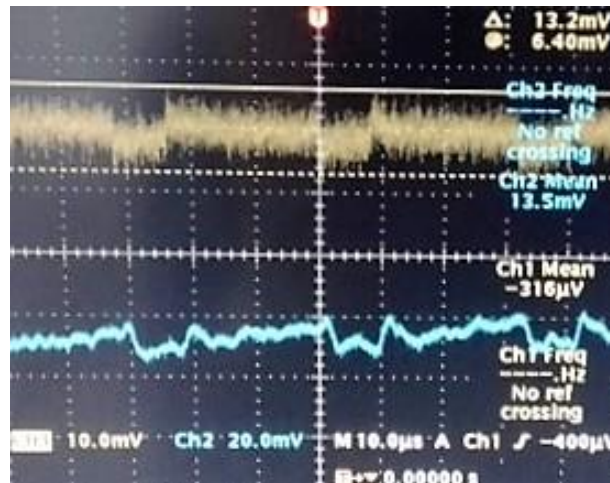


Figure 22. Servo motor low-pass filter test, oscilloscope display.

In yellow (CH1) the filter output voltage. In blue (CH2) the current flowing through the motor (measured with a current clamp, 100 mV/A).

We can see that both signals have strong components at the hash frequency (31 372 kHz), and that the output voltage is disturbed by a high frequency noise. The average output voltage varied from a few hundred μV (idle) to 8.5 mV (blocked) but the noise amplitude was 13.2 mV.

Besides the fact that this measure was unusable (since the noise amplitude was bigger than the measurement range), it was senseless: the cutoff frequency was exaggeratedly low with respect to the theoretical cutoff frequency anticipated. Even with this cutoff frequency, the component at the hash frequency had not been sufficiently diminished.

- Secondly, we sometimes obtained totally incoherent results: we even measured output voltages around 1 and 2 V. This would mean that the current flowing through the motor is about 100-200A, which is impossible.

We tried to determine the origin of these errors. Regarding the high-frequency noise, we think it came from the oscilloscope. Since the output voltage had not been yet amplified, it strongly disturbed the measure. With the MCP6N11, this noise should be negligible enough with respect to the measured voltages.

Concerning the incoherent results, they can be originated in the disturbance signals coming from the circuit (due to the parasitic inductances originated in the wiring, the parasitic capacitance originated in the breadboard etc.). In addition, we believe that having used the oscilloscope to take a differential measure may have distorted the results.

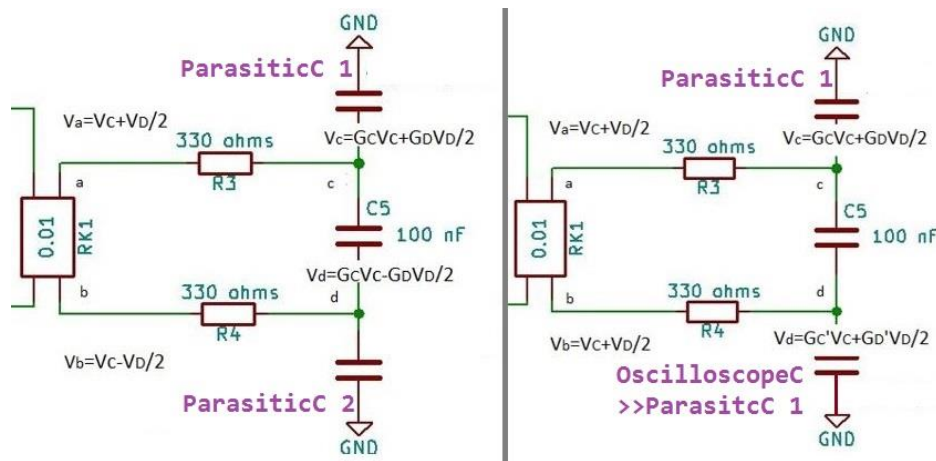


Figure 23. Effect of the parasitic capacitances on the measure of the low-pass filter output voltage.

The problem is illustrated in Figure 23. We want to measure the difference of the voltages in point a ($V_C + V_D/2$, sum of a common component and a differential component) and in point b ($V_C - V_D/2$): that is the differential voltage V_D . In a perfect circuit, the filter is perfectly symmetrical and we can consider that the parasitic capacitances between points c and d and the ground are symmetrical too (these correspond to capacitances *ParasiticC 1* and *ParasiticC 2* on the left diagram). Therefore, the common gain (G_C) and the differential gain (G_D) are equal for points c and d: the filter output voltage is $V_c - V_d = G_D \cdot V_D$, proportional and very close to the voltage we want to measure.

If, however, we use an oscilloscope to take the measure of the voltage between c and d, we will connect its mass to one of these points (to point d, for example) and its probe to the other point (point c). But the oscilloscope mass may be connected to the earth, and the circuit ground too. Since the oscilloscope is a rather big metallic mass and the tested circuit is another metallic mass, connecting the oscilloscope mass to point d can be assimilated to generating a big parasitic capacitance between point d and the ground. In this case, the common and differential gains are different for points c and d: as a result the common component (V_C) skews the measure of $V_c - V_d$.

The solution was to build a PCB. On the one hand, the parasitic capacities and inductances are fewer in a printed circuit than in a test circuit on a breadboard. On the other hand, if we included the amplification circuit we would not have to

use an oscilloscope to take a differential measure: the voltage to measure would be the instrumental amplifier output.

4.2.6. REDUCED PRINTED CIRCUIT AND SECOND TEST PHASE

We decided to make an intermediary PCB, the schematic of this circuit can be found in Annex 1.4. Our purpose was to test the amplification without having to execute the routing of the complete servo motor circuit (this would be done in a further step: it would require a lot of time since the circuit had to be inserted in the servo motor housing).

We expected most of the disturbance signals to disappear with the intermediary PCB. To this end, we gave special attention to the routing of this printed circuit. A ground plane was placed around the filter, in order to isolate it from other signals. Furthermore, we tried to trace a circuit as symmetric as possible around the filter (to avoid non symmetric parasites).

We finally obtained a functional circuit. It was tested several times and, in all cases, the instrumental amplifier output measure was coherent with the intensity measures provided by the external power supply.

Moreover, the noise in the measures had been significantly reduced: with the intermediary PCB, the average output voltages corresponding to a motor running on empty and blocked motor were perfectly distinguishable. This can be observed on Figure 24.

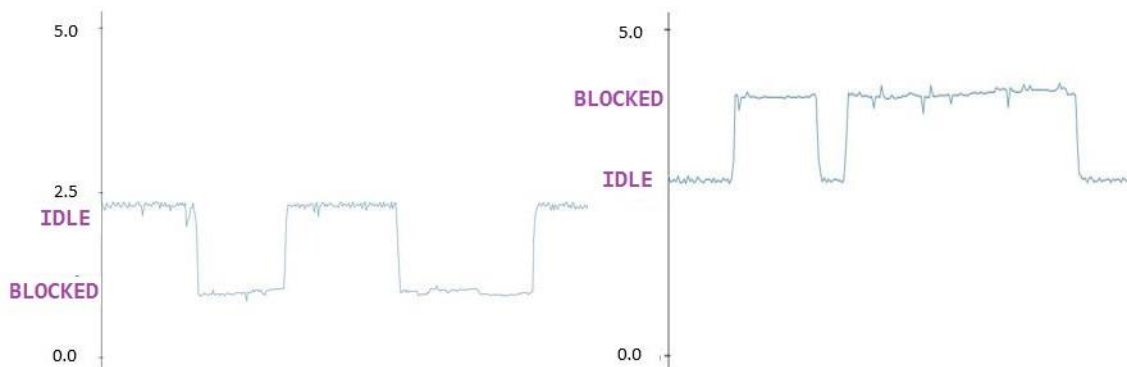


Figure 24. Instrumental amplifier output voltage.

The images correspond to the motor directions of rotation: in the left one, $V_{OUT} \in [0.5, 2.5]$ V so $I_{MOT} \in [-2, 0]$ A; in the right one, $V_{OUT} \in [2.5, 4.5]$ V so $I_{MOT} \in [0, 2]$ A. In addition, when the motor ran with no charge or was stopped V_{OUT} levelled off. In result, both states were perfectly distinguishable. This image was obtained by measuring V_{OUT} with an analog pin of an Arduino Uno board and displaying the result with the Serial Plotter of the Arduino IDE.

4.2.7. FINAL PRINTED CIRCUIT

The greatest difficulty in the creation of the complete printed circuit for the servo motor was its routing. As we have mentioned, it was necessary to adapt the new

circuit to the servo motor housing: this meant creating a PCB of very small dimensions. Besides, special attention needed to be given to the filter to avoid disrupting its signals:

- Once again, we placed a ground plane around the circuit and we tried to keep a symmetric routing from the kelvin resistor to the instrumental amplifier.
- We tried to limit the length of high-current copper traces (the ones going from the H-bridge to the free-wheeling diodes and the motor) in order to limit high-current loops and induced magnetic fields.
- We located these high-current traces as far as possible from the filter on both faces of the PCB and placed ground planes around them.

On Figure 25 we can observe the dimensions of the PCB and where the filter has been placed in it.

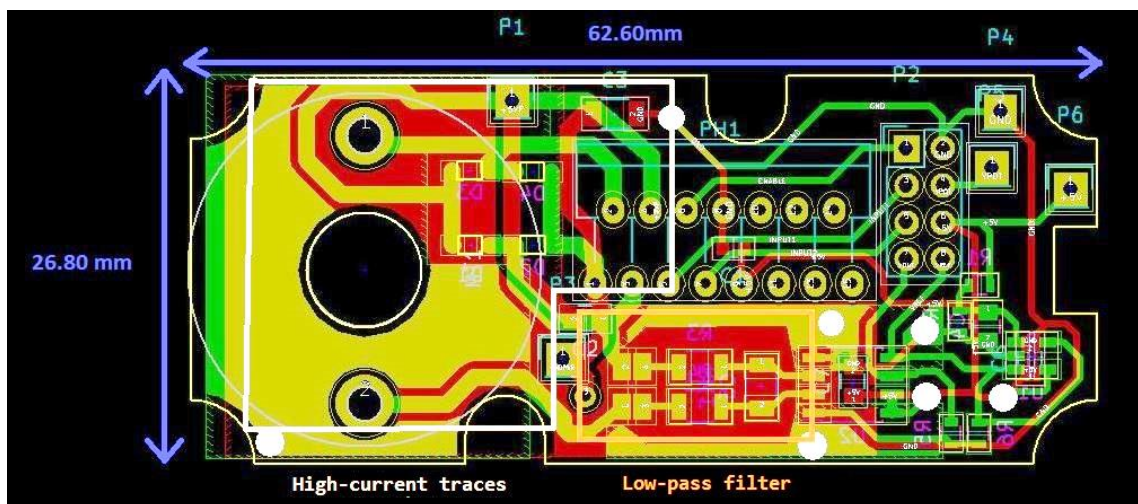


Figure 25. Routing of the complete new servo motor circuit.

4.3. GRIPPER SLAVE CARD (SERVO MOTOR CONTROL AND POWERING CIRCUIT)

Once the conception of the servo motor circuit was completely finished, it was necessary to create another printed circuit to control and power the previous one. The result was a PCB with two independent sub-circuits: one to power the DC motor; another one to command and control the servo motor electronics and to power components other than the motor. The reason why it was decided to integrate both circuits on the same PCB is that it would be easier to install it inside the robot.

For the first sub-circuit, due to the lack of time before the contest, we decided to study and “reuse” the design work developed by another sub-team. More precisely, it was the sub-team that had been in charge of conceiving the general power-supply card (section 2.3). They had later designed another power-supply circuit for a servo motor used on the second robot, which is also a Hitec-805BB. Even if the gripper’s servo motor had been modified, the maximal current

consumption we measured (around 1.5A) coincided with the maximum consumption of the other servo motor.

The central component of the powering sub-circuit is the regulator LM2679, which delivers 5V. These are transmitted with two wires ending in the servo motor circuit.

For the second sub-circuit, it includes the components present in any other slave card (an ATmega328P and its environment, a connector for the system's bus etc., explained in detail in Chapter 2). The inputs and outputs of the servo motor circuit are connected to its microcontroller with an eight-wire ribbon cable.

The schematic of this PCB can be found in the Annex 1.5. It includes a low-pass filter presented in the following section.

4.4. LAST TEST PHASE

Once the final servo motor circuit and the control circuit were ready, a third test phase allowed to test these two cards (separately and together). We detected some faults that were corrected.

Firstly, we discovered that some of the wires in the ribbon-cable connecting the servo motor PCB to the powering and control PCB were defective, and they were substituted.

There was a small noise at the hash frequency in all circuit signals. This was because the previewed decoupling capacitor (of 22 μ F, presented on section 4.2.4) was not big enough. We substituted the latest by a capacitor of 47 μ F and the noise was considerably reduced.

We also found that the instrumental amplifier output voltage was disturbed by several noises of very different frequencies. We thought that one of the main sources of these noises was also the ribbon-cable (in it, several long cables delivering different signals are packed together). Therefore, we decided to place another first-order low-pass filter between the ribbon-cable connector and the ATmega328P (in other to get a proper signal V_{OUT}). After adding this low-pass filter (cutoff frequency of 1 kHz), we observed:

- A noise signal of 400 Hz, the origin could not be determined.
- A noise signal of, approximately, 260 kHz. This was generated by the switch mode regulator used to power the servo motor. In this regulator, there is a transistor commuting at 260 kHz. This generates highly variable electromagnetic fields that affect the servo motor circuit. With the additional low-pass filter, the noise amplitude decreased, but not enough (from 5V spikes to 1.3 V spikes).

These disturbances were solved by computing, with programming, the average of V_{OUT} . With it, the peaks disappeared, allowing to perfectly distinguish a blocked motor from an idle motor.

4.5. GRIPPER CODE

The code to control the gripper has been conceived using OOP and implemented on the slave gripper PCB. We find three classes representing the elements of the subsystem: *Potentiometer*, *Servomotor* and *Gripper*.

- The first one allows to measure the voltage between the terminals of a potentiometer (to measure the motor rotation angle in this case).
- The second one, *Servomotor*, corresponds to the modified Hitec-805BB: with it, we can command the servo motor rotation and measure the current flowing through it.
- *Gripper* represents the entire gripping subsystem. It includes an object *Potentiometer* and an object *Servomotor*. Thanks to its methods (as *goToAction()*), used by the gripper's main program, the master can control the whole subsystem by sending simple instructions through i2c.

The details of this code can be found in Annex 2.2.

4.6. CONTEST RESULTS AND IMPROVEMENTS

The gripping subsystem electronics worked as expected during the contest. However, we sometimes had problems with the mechanics: the slit in the metal plate, guiding the movement of the gripper, was not perfectly straight. In result, the gripper sometimes got stuck. The solution was to grease the linkage, what we did during the contest.

It is however important to point out that, during the contest, this system was not used as much as we intended to. We experienced many problems concerning both robots: we then concentrated on solving them and did not have enough time to adapt the robot's actions to the different possible seashell configurations. The solution was to make the robot search only the seashells next to the area of departure (which, in all configurations, were neutral or of the team's color).

A possible improvement for the system would be to create a controller to command the current flowing through the servo motor when the gripper grabs an object. For the moment, when we order the gripper to grab an object, the voltage that is applied to the motor has been determined empirically (it corresponds to the force required to seize a seashell). It could be interesting to create a controller to directly command the intensity flowing through the motor (proportional to the applied torque) and, afterwards, a program to dynamically adapt the intensity to the gripper's load.

CHAPTER 5: THE COLOR-SENSING SUBSYSTEM

5.1. INTRODUCTION

At the beginning of the year, two options were taken into consideration to resolve the seashell configuration problem (section 1.2.3). Briefly, in order to choose the seashell configuration of each match (the ways seashells are distributed throughout the playing field), we could either configure the robot before the match or develop a color-sensing system. This would be integrated in the gripper, letting the robot sense the color of the seashells.

Initially, as we had limited time before the contest, we decided to implement the first option, via a PCB conceived by another member of the team. It included several buttons and a LCD screen that allowed to preconfigure the robot and to display orders to prepare it correctly before each match. However, we decided to develop the color-sensing subsystem at the end of the academic year, once the contest had finished. The main reason for this choice was its pedagogical interest: we estimated that it would be very useful to learn the main issues of light sensing technologies and that it was a good task to complete the project.

At the beginning of the year, we started studying two specific color sensors found at the LISA: the TCS3200 [7] and the APDS9960 [8]. We finally decided to develop our system with the first one, as it is a model often used at the LISA in student projects. There were then many units available and the reports of these projects could be very useful sources of information.

5.2. THE TCS3200

The TCS3200 is a light-to-frequency converter. It is composed by 64 photodiodes: 16 have blue filters, 16 red filters, 16 green filters and 16 have no filter (clear photodiodes). They are disposed in an 8x8 matrix, interspersed.

The component works as follows. Two input pins (S2, S3) let us choose which photodiodes are activated (blue, red, green or clear). The output pin gives a square signal of frequency proportional to the irradiance (power received by a surface per unit area) of the incident light observed with the chosen filter. The set of outputs obtained with the four filters give useful information about the spectrum content of the light the sensor receives.

For example, if the TCS3200 is illuminated with light of wavelengths near the red color in the visible spectrum, the irradiance measured with the red-filtered photodiodes will be higher than with the blue or the green-filtered photodiodes.

Two other pins (S0, S1) allow to scale the output frequency.

5.3. DESCRIPTION OF THE PROBLEM

In our case, we were interested in distinguishing the color of different objects. More specifically, the color of the mobile objects in the playing field: seashells (purple, green and white) and sand blocks (orange), since they can eventually be seized by the gripper.

A regular source emits light of many different frequencies of the visible spectrum. When an object is illuminated, it absorbs some of these frequencies and reflects others. These last frequencies are the ones that arrive to our eyes and are responsible for the color we perceive (e.g. if the reflected frequencies correspond to the red color in the visible spectrum, we will see a red object).

The task was then to analyze the light reflected by the objects in the field.

5.4. FIRST TEST PHASE

We decided to start by simply connecting a TCS3200 to an Arduino Uno, to get used to its functioning. The output signal was analyzed using an oscilloscope.

This first test phase served to extract one conclusion: the ambient light conditions, highly variable, disturb the measures of the color sensor. We were able to see how the TCS3200 output frequency changed as we approached objects of different colors, which was expected, but this frequency fluctuated strongly with slight changes in ambient light. For example, if a person approaches the sensor and a light shadow appears on it, its output changes.

It was then concluded that we would need to isolate the sensor from ambient light, and to illuminate the objects with a specific light source.

5.5. PROTOTYPE CONCEPTION

It was then necessary to create a prototype of the color-sensing system to continue with the tests (containing the TCS3200, a source of light and a barrier to block ambient light). The reading of several reports and websites [29], as well as some other tests that we performed, made us realize that several factors needed to be taken into account to create a satisfying prototype:

- As it has been mentioned, the light sensor has to be isolated from ambient light.
- Because of this, the observed surface has to be illuminated. The most convenient source are LEDs, since they can be easily integrated in a PCB.
 - However, the light coming from the LEDs must not directly illuminate the TCS3200 (what can lead to erroneous measurements). All or most of the light arriving to the sensor must be reflected from the observed object.
- The relative position between the LEDs, the sensor and the observed object must be constant. The output signal frequency depends on the irradiance of the different light frequencies arriving to the sensor. Irradiance

(Power/Surface) diminishes if the distance of the source of light to the sensor increases. If the distances between the source of light and the observed object and between the observed object and the sensor vary from one measure to another, results may not be coherent.

Taking all these aspects into consideration, we designed a prototype composed of a printed circuit board and a case mounted on it. We can observe it on Figure 26.

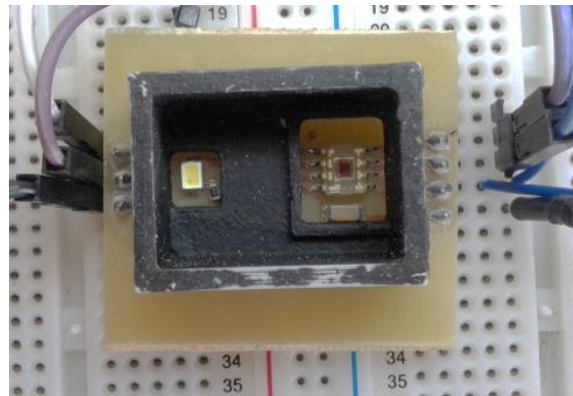


Figure 26. Prototype of the color-sensing system.

The PCB.

The schematic of the printed circuit board is very simple. It mainly contains a TCS3200 and a LED. All the TCS3200 inputs and outputs, the 5V and GND tracks are connected to headers. These can be connected to an Arduino board or other circuit through wires.

We chose a white led, multicom 703-1026 [30]. White LEDs are the most convenient source of light for this application, since the light they emit covers almost all the visible spectrum (almost all colors can be visualized). However, it is important to note that there are no ideal white LEDs (that emit the same light intensity for all frequencies).

The resistor in series with the white LED was dimensioned to obtain 2 cd of luminous intensity (later on, tests showed that this was enough to illuminate the objects and small enough to not saturate the light sensor).

The case.

We can observe the case on Figure 27. The bottom is mounted on the PCB. The upper opening is where the observed object is placed, isolating the inside from external light. A border on the hole corresponding to the TCS3200 prevents the LED light to directly illuminate the sensor.

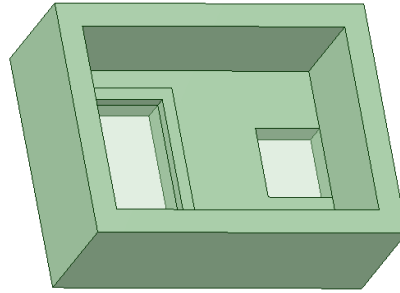


Figure 27. Color-sensing system prototype case.

It is a case of very small dimensions (26.5x20x10 mm). This is because the final color-sensing system would be installed on the robot gripper. Therefore, we wanted to create a prototype as similar as possible to the final system (to be able to extrapolate the results of the prototype tests to the final system). However, with these small dimensions, it is sure that some of the incident light on the sensor comes from the walls of the case and not from the observed object. To avoid erroneous measurements, the case was painted in matte black (ideally, it absorbs all frequencies of the visible spectrum).

The schematic of the PCB is included in the Annex 1.6.

5.6. SECOND TEST PHASE (PROTOTYPE)

Once the prototype was finished, we proceeded to test it to verify it work as expected.

Once again, we used an Arduino Board to control the PCB and an oscilloscope to measure the frequency of the output signal. We used cardboards of seven different colors to try the prototype.

We can find the results of these tests in

Table 1.

CARDBOARD COLOR	FILTER				
	CLEAR	RED	BLUE	GREEN	RED + BLUE + GREEN
PINK	212	119	49	52	220
PURPLE	168	92	46	40	178
BLUE	138	49	46	45	140
GREEN	133	47	32	55	134
LIGHT GREEN	145	63	25	59	147
YELLOW	246	134	36	82	252
ORANGE	178	118	25	45	188
RED	104	78	19	19	116

Table 1. Color-sensing system prototype first tests results: TCS3200 output frequencies (All frequencies in kHz, full-scale frequency : S0=1, S1=1)

In the previous table, we can observe the TCS3200 output frequencies for seven different colors (listed in order of their spectral frequency), each color being measured with the four existing filters. We decided to round the oscilloscope measures to three significant digits (units, tens and hundreds for the measures in kHz, since the first decimal digit value fluctuated in all cases).

We can observe that the results for each color are not comparable. This is probably because, as it has been mentioned, the emission spectrum of the used LED is not perfect (shown in Figure 28).

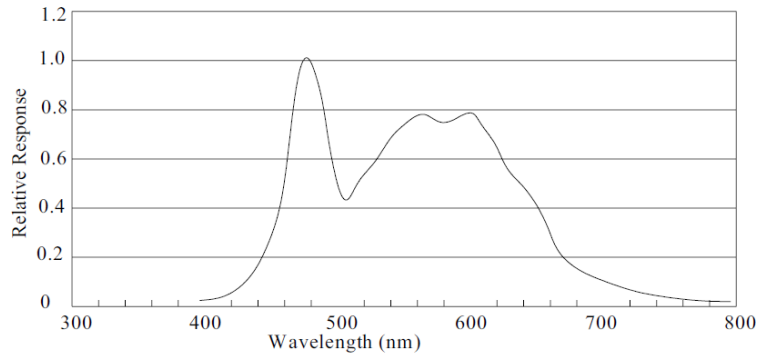


Figure 28. Emission spectrum of the white LED multicomponent 703-1026 [30].

With this spectrum, we can expect for the total irradiance of wavelengths around 500 nm (blue, green) or bigger than 650 nm (red) to be smaller than for the rest of wavelengths. Precisely, total irradiance can be measured with the TCS3200 output frequency when no filter is applied (column “CLEAR”). Also, with the sum of the TCS3200 output frequencies when red, blue and green filters are applied (column “RED + BLUE + GREEN”, we can observe in

Table 1 that the values of this column are very similar to the values of “CLEAR”).

The previous hypothesis was verified: the values of “CLEAR” and “RED + BLUE + GREEN” were smaller for blue, green and red than for the rest of the cardboard colors.

Therefore, to compare the measures taken with the red, blue and green filters for different objects, we need to normalize them: they can either be divided by the value of “CLEAR” or by “RED + BLUE + GREEN” of the corresponding object. The normalized values for the measures in

Table 1 are presented in Table 2.

CARDBOARD COLOR	(* / CLEAR) · 100			(* / (RED + BLUE + GREEN)) · 100		
	RED	BLUE	GREEN	RED	BLUE	GREEN
PINK	56	23	25	54	22	24
PURPLE	55	27	24	52	26	22
BLUE	36	33	32	35	33	32
GREEN	35	24	41	35	24	41
LIGHT GREEN	43	17	41	43	17	40

CARDBOARD COLOR	(* / CLEAR) · 100			(* / (RED + BLUE + GREEN)) · 100		
	RED	BLUE	GREEN	RED	BLUE	GREEN
YELLOW	54	15	33	53	14	33
ORANGE	66	14	25	63	13	24
RED	75	18	18	67	16	16

Table 2. Normalized results of the color-sensing system prototype first tests.

5.7. CODE TO MEASURE THE FREQUENCY

After having successfully built a prototype, we needed to find a way to measure the TCS3200 output frequency with a microcontroller. As the slave cards of the robot contain ATmega328P microcontrollers loaded with the Arduino Uno bootloader, we could consider using Arduino functions. Among these, *pulseIn()* [31] seemed very convenient for our purposes.

pulseIn() reads a pulse on an I/O pin and returns its length in microseconds. It also allows to specify a *timeOut* (if the pulse length exceeds *timeOut*, the function returns 0).

This function could then be used to measure the half-period of the color sensor output signal, to later convert this measure into a frequency. But the Arduino documentation specifies two important facts: *pulseIn()* works with lengths between 10 and 180 000 000 microseconds ($f \in [0.00278, 50000]$ Hz) and its resolution is better with short pulses.

TCS3200 pins *S0* and *S1* allow to scale the output frequency. It is however impossible to know in advance the exact range of the measured frequencies (it depends on the test conditions and on the colors measured). We only know the typical saturation frequency of each configuration (600 kHz, 120 kHz and 12 kHz). With the first two, it is possible for the measures to exceed 50 kHz (limit of *pulseIn()*). However, it is more efficient to measure high frequencies (better resolution of *pulseIn()* and faster measures). We then tried each of the configurations to choose the most appropriate.

- **Configuration 1** (typical $f_{SAT} = 600 \text{ kHz}$). The results of the first tests (table 1) had already shown that the measures largely surpassed 50 kHz.
- **Configuration 2** (typical $f_{SAT} = 120 \text{ kHz}$). We used some colored cardboards and the objects of the playing field to test it. Almost all measures were under 50 kHz, except for white seashells.

Period measures were also taken, using an oscilloscope, and were later compared to the values returned by *pulseIn()*. We could observe that *pulseIn()* imprecision increased when 50 kHz were surpassed. We decided then to discard this configuration.

- **Configuration 3** (typical $f_{SAT} = 12 \text{ kHz}$). We can observe some of the frequency measures realized with this configuration in
- Table 3. All were highly under the *pulseIn()* saturation frequency (50 kHz) and the TCS3200 saturation frequency. It was, then, the only configuration that could be used.

OBJECT COLOR	FILTER				
	CLEAR	RED	BLUE	GREEN	RED+BLUE+GREEN
GREEN (SEASHELL)	0,97	0,30	0,25	0,43	0,98
PURPLE (SEASHELL)	1,12	0,61	0,30	0,27	1,18
WHITE (SEASHELL)	5,10	2,53	1,12	1,62	5,26
ORANGE (SANDBLOCK)	2,43	1,48	0,33	0,70	2,52

Table 3. TCS3200 output frequencies with configuration S0=0, S1=1: $f_{SAT}=12 \text{ kHz}$ (All frequencies in kHz).

All measures, in kHz, were taken with an oscilloscope and rounded up to three significant digits.

Once the TCS3200 configuration was selected, we created a function to obtain a frequency in kHz from a half-period in μS :

```
float frequency(unsigned long demiPeriod){
    return (float)(500.0/(demiPeriod));
}
```

We tested the measures obtained by the combined use of *frequency()* and *pulseIn()*, by comparing them to the ones given by the oscilloscope. We could observe that they differed in the last significant digit (by two or three units). This was because the lengths returned by *pulseIn()* did not perfectly match the half-periods measured with the oscilloscope. However, it was decided that both functions would be used in our system (we would later analyze if these imprecisions impeded distinguishing colors).

5.8. CODE TO DISCERN COLORS

Once we were able to successfully measure the color sensor output signal using a microcontroller, the next step was to develop a programming logic to identify different colors. In general, in all applications using the component TCS3200, this is done by one of the following ways:

- By comparing, for each observed object, the magnitude of the measured irradiance with each of the sensor's filters. That is, to determine the "red content", the "blue content" and the "green content" of an object and to discern its color by comparing them. The TSC3200 spectral responsivity characteristic (Figure 29) can be used to determine how these values should be compared, although the resulting method should always be adjusted and validated with empirical tests .

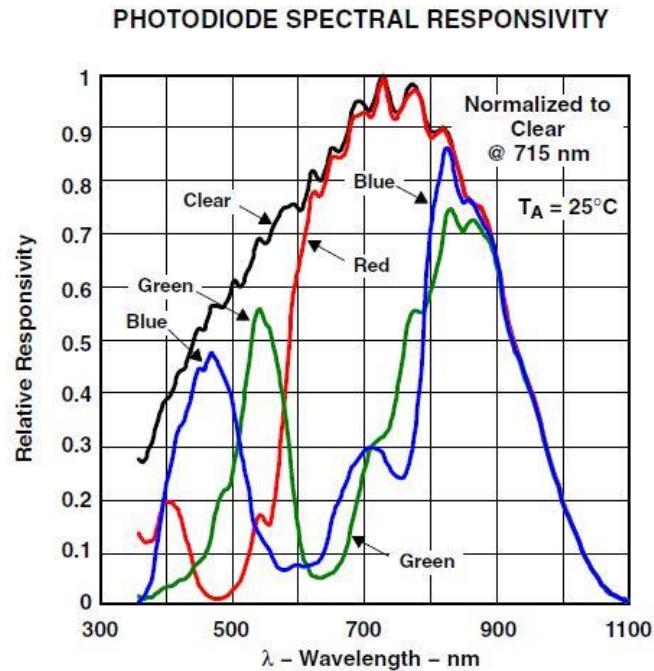


Figure 29. TCS3200 photodiodes spectral responsivity [7]

For example, if the dominant color of a surface is blue and its second dominant color is red, we can determine the surface is violet. But if the second dominant color is green, we can say it is a blue surface.

This logic has a great disadvantage. With it, we can easily distinguish basic colors (e.g. violet, blue, green, yellow and red), but as the color palette augments (we include other colors as orange, white, pink or light green), the comparisons become increasingly complex.

- By creating a vector system. Once the measured frequencies are normalized (as indicated in section 5.6), each observed color can be considered as a vector of three components (RED; BLUE; GREEN). We can create a program in which some pre-defined colors (or vectors) are stored in memory. If we want to identify an object's color, we will measure it with the TCS3200 filters, compute its vector components and compare them to the vectors in memory. For that, the Euclidian distance of the new vector to all the stored vectors can be computed: we will identify the object's color with the color of the nearest vector in memory.

This second option is then simpler than the first one. Storing a new color in memory is equivalent to defining a new vector. The complexity of the logic does not change, no matter the number of colors in memory: only the number of distances computed and compared will increase.

The second option was therefore chosen. To implement it, we used object oriented programming. We created a class *Color* to represent each of the vectors of a defined color system. The method *distance()* allows to easily compute the Euclidian distance to an instance of class *Color*.

A simple Arduino sketch was programmed to test this class, in combination with the functions *frequency()* and *pulseIn()*. The frequencies corresponding to the colors of the playing field objects were measured and these colors were introduced in memory. It is important to note that those frequencies were different than the ones included in

Table 3. This is due to the fact that we had already integrated the color-sensing prototype in the gripper: its case's form had changed and, therefore, the TCS3200 output measures too.

The results were very satisfying: the system could distinguish the four colors (white, green, purple, orange) to perfection. The codes are included in Annex 2.3.

5.9. INTEGRATION IN THE GRIPPER

Two integrations needed to be performed in order to complete the color-sensing system: the mechanical integration and the code integration.

5.9.1. MECHANICAL INTEGRATION

In order to install the color-sensing system, a new gripper was created. It was presented in section 4.1.1. This new gripper maintained the shape of the gripper taken to the contest (that perfectly matched the shape of a seashell), but we perforated a hole in one of its fingers and mounted the color-sensing system on it (as shown in Figure 30). We were able to reuse the prototype for the final color-sensing system, only the case's shape needed to be adapted to the gripper finger's shape: the two objects were then glued impeding ambient light to arrive to the TCS3200, since their shapes matched.

In addition, silicone sheets were placed on both fingers to improve the contact forces between the gripper and its load. Finally, foam rubber was added between the fingers and the silicone sheets (to more easily adapt to the shape of objects other than the seashells and avoid ambient light entering in the color sensor cavity).

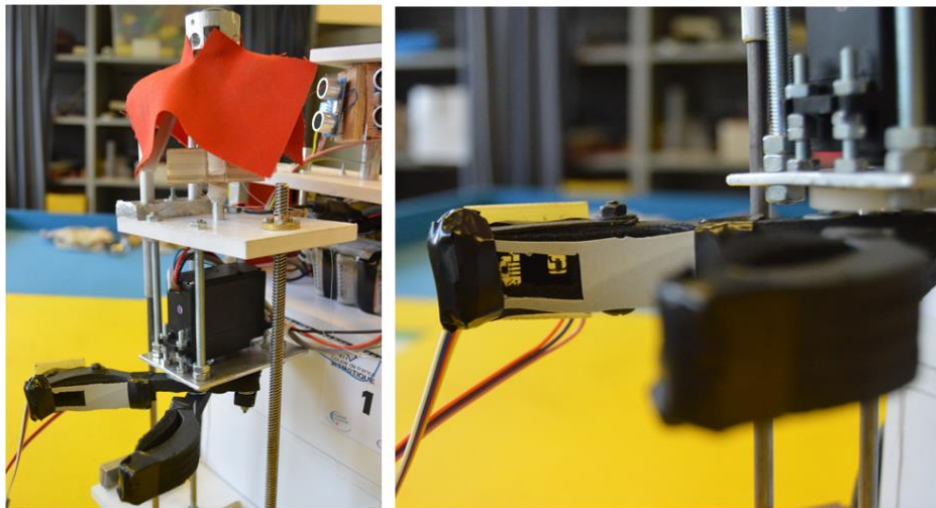


Figure 30. Mechanical integration of the color-sensing system in the gripper.

5.9.2. CODE INTEGRATION

We decided to command the color sensor with the same microcontroller used to command the gripper's servo motor. In the end, the color sensing functionality is complementary to the gripper's functions: it can only be commanded after we order the gripper to grab an object.

Therefore, we completed the code corresponding to the slave gripper to include this new functionality. The class *Color*, presented in the previous section, was included, as well as the new class *Taos* (representing the sensor TCS3200). The latter allows to save in memory several colors in order to identify them afterwards. The class *Gripper* and the main program have been modified to include and use these new classes. As a result, the master can send a new order (*READ_COLOR*) to the gripper and, when requested, it receives information about the red color.

All files containing these modifications are included in the Annex 2.4 (only the modified files or the new files with respect to Annex 2.3 have been included).

5.10. RESULTS

The extended gripping subsystem was tested and worked as expected. The only unexpected event was that, sometimes, it did not manage to grab the objects properly. In these cases, the hole on its finger was not completely blocked and some ambient light arrived to the light sensor. However, even in such cases, the system always detected the right color.

CONCLUSIONS

At the beginning of the year, we set ourselves the goal of accomplishing a module to perform the tasks “Seashell Collecting” and “Funny Action” before the Eurobot in May 2016. Despite the problems encountered during the contest, this objective has been fulfilled: we achieved a functional system that allowed the robot to perform tasks and score points.

Having the robotics contest as a target has turned the conception of this module into a race against the clock, and this has added great value to the project. It has not been possible to develop all the ideas we had or to make the system meet all the sub-goals we initially defined. Indeed, it has been necessary to prioritize. From this, we have learnt how important it is to build a solid and safe system before considering improving it.

However, the ultimate aim of this project (and of the Eurobot contest) is to help students develop their knowledge and skills in electronics and other areas. This objective has been largely achieved: we have had the opportunity to work on servo motor and stepper motor control, to develop a color sensor or to design several mechanical systems, among other things. In addition, during the year, pedagogical interest has been a criterion often used to choose the systems that were implemented. Finally, even when the contest was finished, we have tried to improve and complete our work.

Moreover, even if this project had a major pedagogical interest, the technologies that have been developed and the knowledge that has been acquired are truly useful in numerous industrial projects nowadays.

REFERENCES

- [1] Planète Sciences, “Eurobot. The beach bots – 2016. Rules”, September 2015,
http://www.eurobot.org/attachments/article/51/E2016_Rules_EN.pdf
- [2] E. Brown, N. Rodenberg, “Universal robotic gripper based on the hamming of granular material”, September 2010,
<http://www.pnas.org/content/107/44/18809.full.pdf>
- [3] Institut Maupertuis, “ROBOTIQUE: les Préhenseurs Adaptatifs”, Avril 2014,
http://www.institutmaupertuis.fr/include/telechargement.php?id_doc=142&fichier=1
- [4] M. Fässler, “Force Sensing Technologies”, EHT, Spring Term 2010
http://students.asl.ethz.ch/upl_pdf/231-report.pdf
- [5] Gotronic, Force Sensing Resistors,
<http://www.gotronic.fr/pj-492.pdf>
- [6] Gotronic, Micro load cell CL635 datasheet
<http://www.gotronic.fr/pj-459.pdf>
- [7] AMS, TCS3200 color sensor datasheet
<http://ams.com/eng/content/download/250259/976005/142755>
- [8] Avago Technologies, Digital proximity ambient-light RGB and gesture sensor datasheet
<http://www.avagotech.com/products/optical-sensors/integrated-ambient-light-and-proximity-sensors/apds-9960>
- [9] RaspberryPi, Raspberry Pi 2 model B
<https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>
- [10] Fairchild, LM78XX positive voltage regulator datasheet
<http://www.fairchildsemi.com/datasheets/LM/LM7805.pdf>
- [11] Texas Instruments, LM2679 simple switcher 5-A step-down voltage regulator with adjustable current limit datasheet
<http://www.ti.com.cn/cn/lit/ds/symlink/lm2679.pdf>
- [12] Arduino, “Building an Arduino on a breadboard”, October 2008
<https://www.arduino.cc/en/Main/Standalone>
- [13] Atmel, 8-bit Microcontroller with 4/8/16/32 kB in-system programmable flash datasheet
http://www.atmel.com/images/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf

[14]Noguét, “GUIDER EN TRANSLATION : la liaison glissière”, Lycée Blaise Pascal, Colmar, April 2009

<http://sitemecanique.free.fr/Documents%20TGE1/Etude%20des%20constructi ons/Technologie/CI6%20Analyser%20et%20concevoir%20la%20solution%20co nstructive%20d'une%20liaison/Liaison%20a%201%20degre%20de%20liberte/C ours%20Guidage%20en%20translation.pdf>

[15]NYU Polytechnic School of Engineering, “Static and kinetic friction”,
http://engineering.nyu.edu/gk12/Information/Vault_of_Labs/Physics_Labs/static%20and%20kinetic%20friction.doc

[16] “Systèmes vis-écrou pour transmission de puissance”,
http://58consmeqa.free.fr/Cours%20a%20Pdf/vis_ecrou.pdf

[17] Mclennan Servo Supplies Ltd., 42DBL series digital linear actuators datasheet,

<http://docs-europe.electrocomponents.com/webdocs/0625/0900766b806255a6.pdf>

[18] Texas Instruments, ULN200x ULQ200x high-voltage high current darlington transistor arrays datasheet,

<http://www.ti.com/lit/ds/symlink/uln2003a.pdf>

[19] Pololu Electronics, Stepper Motor with 28 cm Lead Screw,
<https://www.pololu.com/product/2268>

[20] Texas Instruments, DRV8825 Stepper motor controller IC,
<http://www.ti.com/lit/ds/slvsa73f/slvsa73f.pdf>

[21] Pololu Electronics, DRV8825 Stepper Motor Driver Carrier,
<https://www.pololu.com/product/2133>

[22] Fairchild, MMBT2222 NPN general purpose amplifier datasheet,
<https://www.fairchildsemi.com/datasheets/MM/MMBT2222.pdf>

[23] Fairchild, KSA1010 PNP epitaxial silicon transistor datasheet,
<https://www.fairchildsemi.com/datasheets/KS/KSA1010.pdf>

[24] Fairchild, KSC2334 NPN epitaxial silicon transistor datasheet,
<https://www.fairchildsemi.com/datasheets/KS/KSC2334.pdf>

[25] ST, L298 dual full-bridge driver datasheet,
<http://www.st.com/content/ccc/resource/technical/document/datasheet/82/cc/3f/39/0a/29/4d/f0/CD00000240.pdf/files/CD00000240.pdf/jcr:content/t ranslations/en.CD00000240.pdf>

[26] Multicomp, MBR30xxCT diode schottky datasheet,
<http://www.farnell.com/datasheets/1697458.pdf>

[27] Infineon, BAT60A silicon schottky diode datasheet,
http://www.infineon.com/dgdl/Infineon-BAT60ASERIES-DS-v01_01-en.pdf?fileId=db3a304313d846880113def70c9304a9

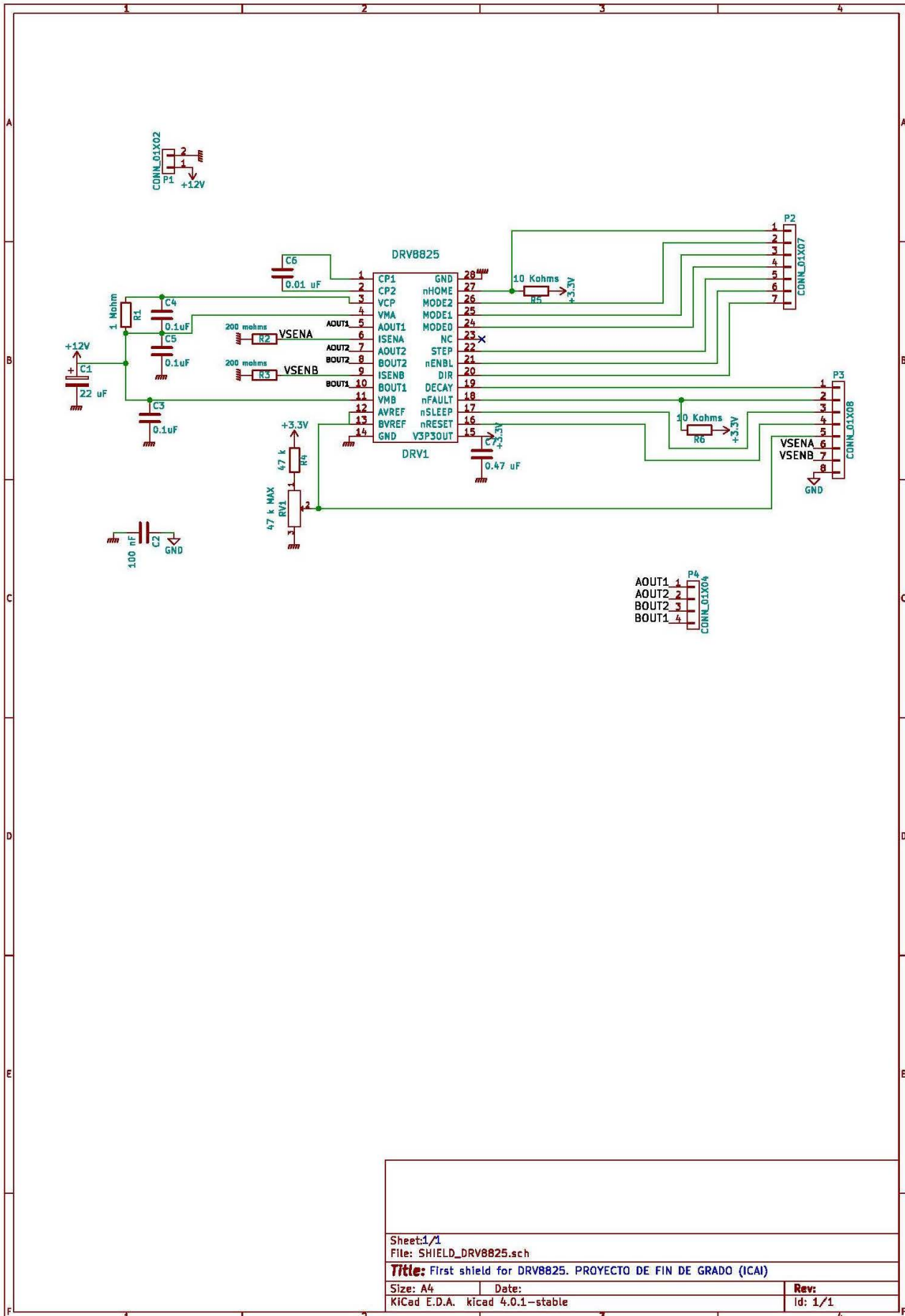
- [28] Microchip, MCP6N11 500 kHz 800 μ A instrumentation amplifier datasheet,
<http://ww1.microchip.com/downloads/en/DeviceDoc/25073A.pdf>
- [29] S. Iooss, C. Dancette, A. Rozier. “Rapport de l’activité expérimentale : étude d’un capteur de couleur”, École Centrale Paris, March 2015
- [30] Multicomp, 703-1026 white led datasheet,
http://www.farnell.com/datasheets/1636566.pdf?_ga=1.113184900.1018934932.1459804315
- [31] Arduino, `pulseIn()` reference,
<https://www.arduino.cc/en/Reference/PulseIn>
- [32] Ericsson Components, “Industrial Circuits Application Note: Stepper Motor Basics”, Stepping modes,
<http://www.solarbotics.net/library/pdflib/pdf/motorbas.pdf>
- [33] Hitec, HS-805BB+ mega 1/4 scale servo datasheet
<http://www.robotshop.com/media/files/pdf/hs805.pdf>
- [34] Futaba, S3003 servo datasheet
http://mech.vub.ac.be/teaching/info/mechatronica/finished_projects_2016/Wire_Guidance/images/et-servo-s3003.pdf

PART II

ANNEXES

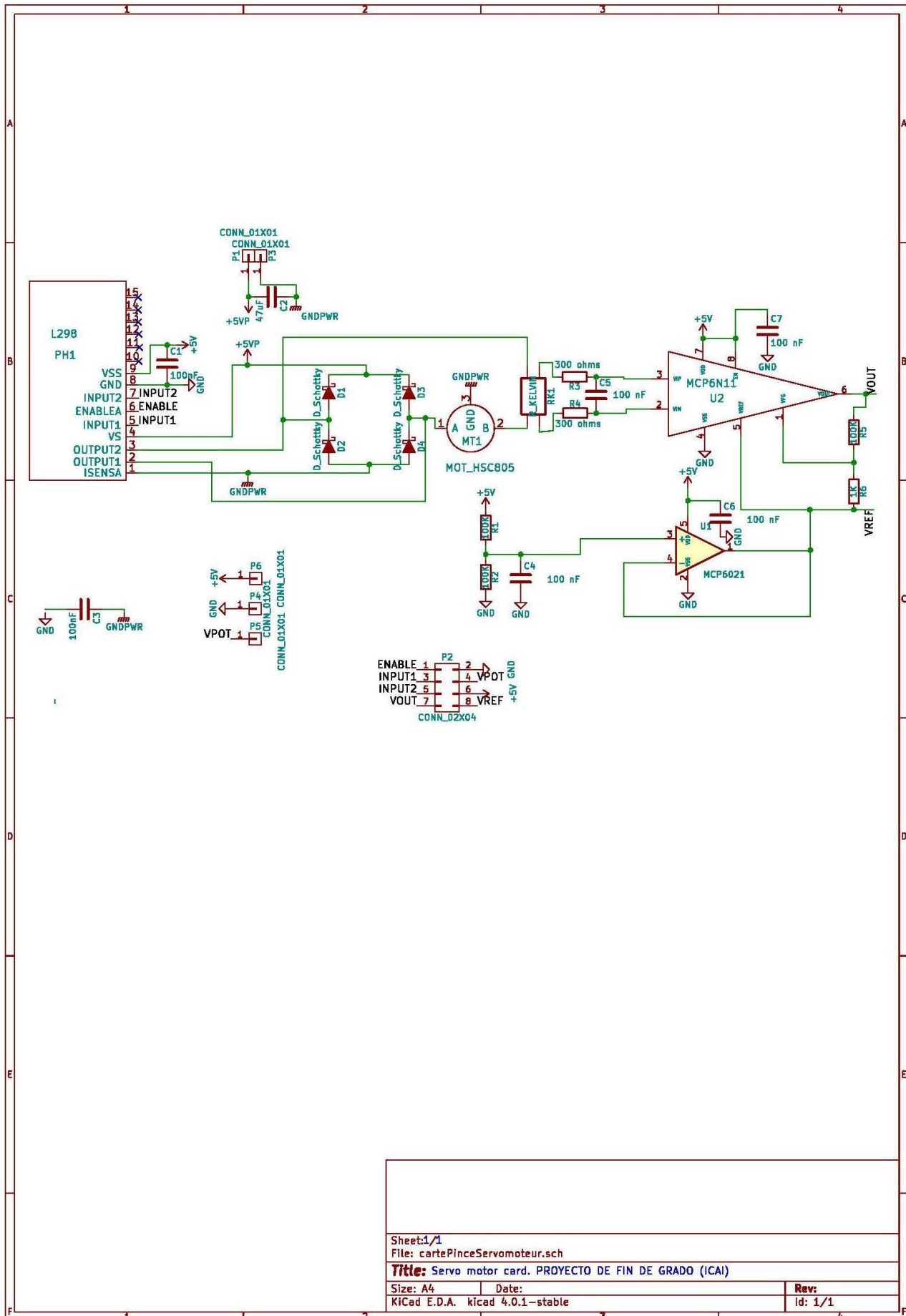
ANNEX 1: PRINTED CIRCUIT BOARD SCHEMATICS

1.1. FIRST SHIELD FOR DRV8825

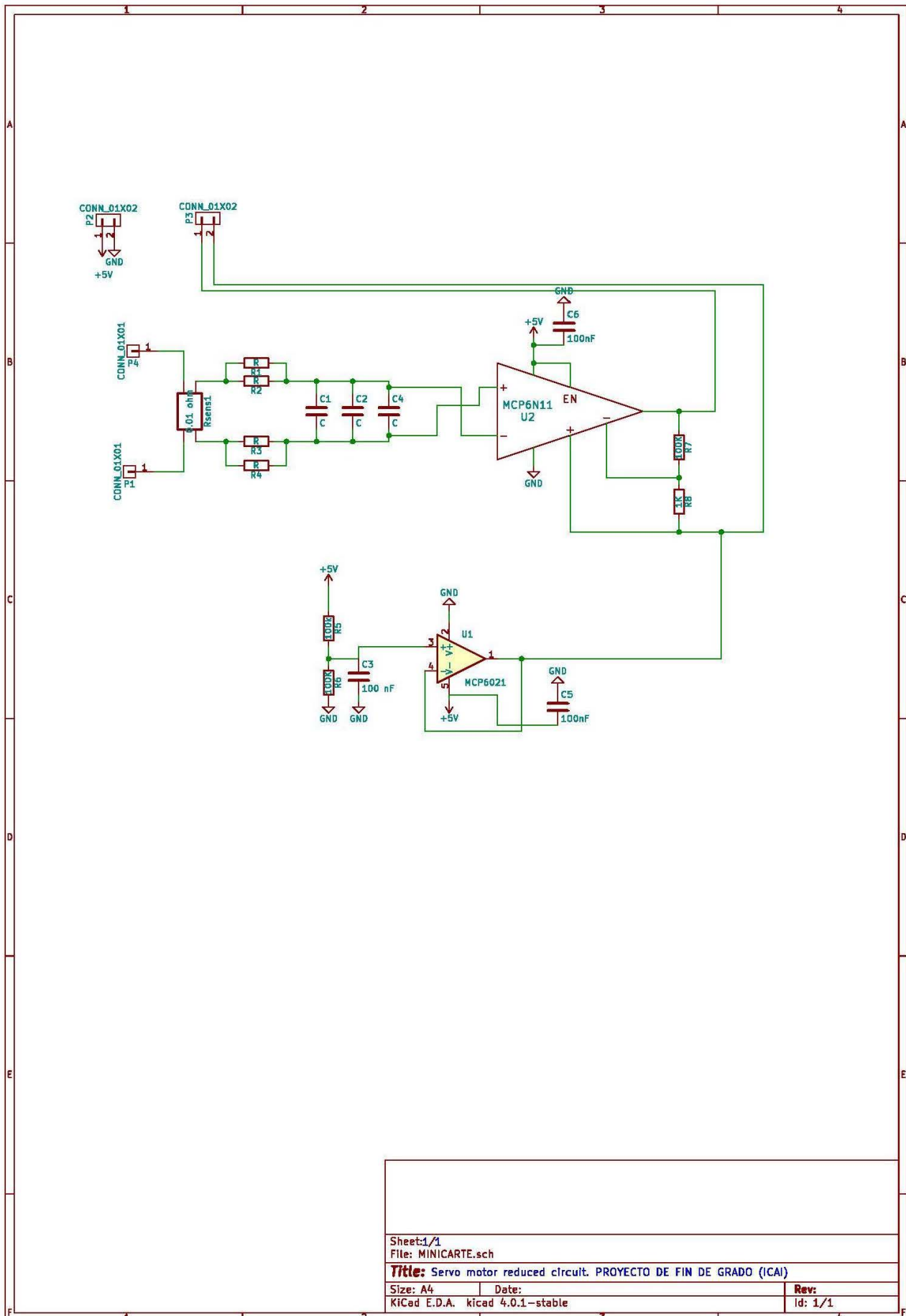




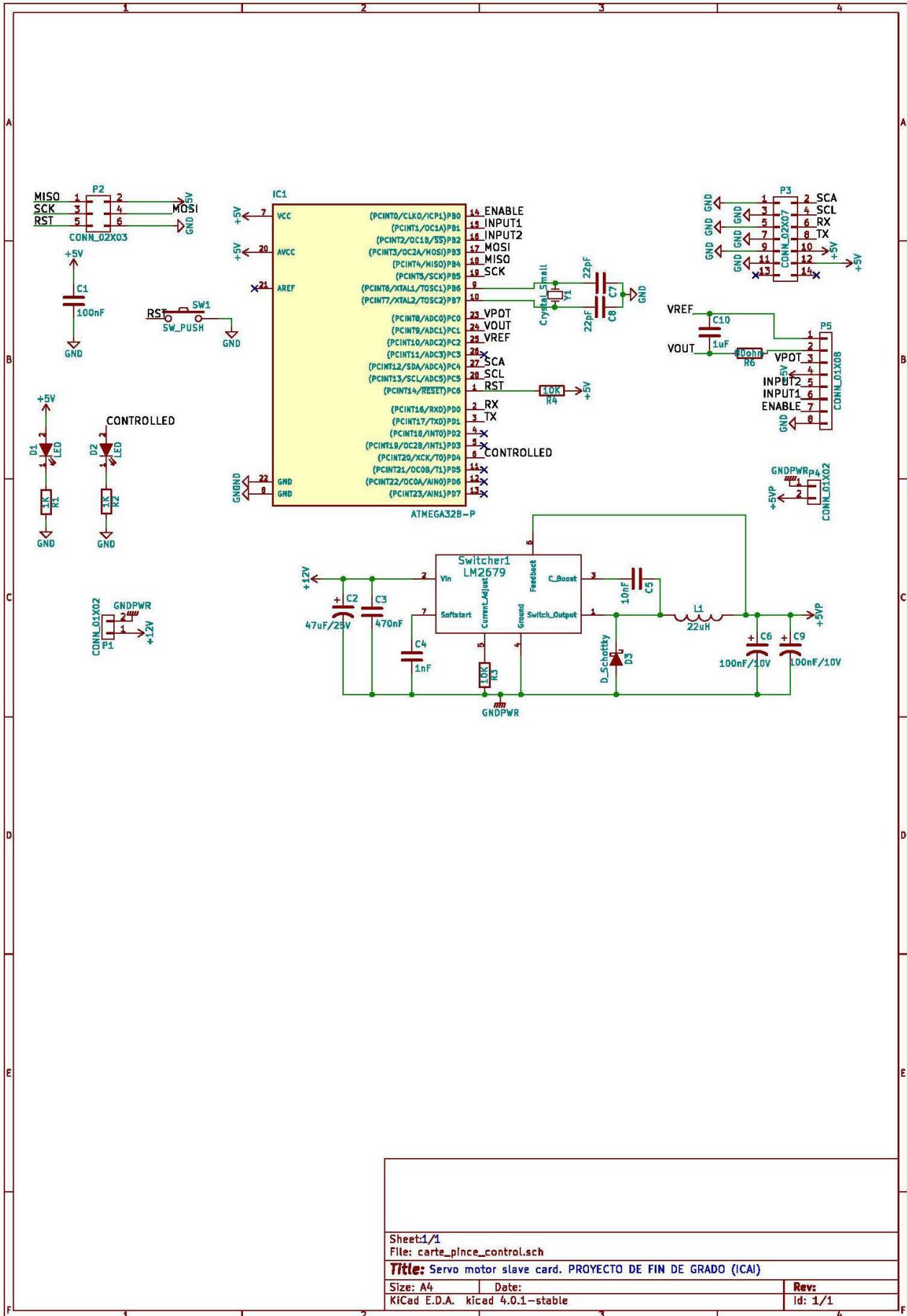
1.3. SERVO MOTOR CARD



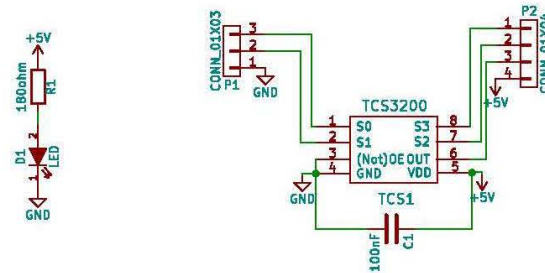
1.4. SERVO MOTOR REDUCED CIRCUIT CARD



1.5. SERVO MOTOR SLAVE CARD



1.6. COLOR -SENSING SYSTEM PROTOTYPE CARD



Sheet: 1/1
 File: prototypeCapteurCouleur.sch
Title: Color sensing system prototype card. PROYECTO DE FIN DE GRADO (ICAI)
 Size: A4 Date: Rev:
 KiCad E.D.A. kicad 4.0.1-stable Id: 1/1

ANNEX 2: CODE

2.1. SLAVE ELEVATOR

```
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include "Elevator/Elevator.h"
4
5
6
7 //CONSTANTS
8 #define STEP_PIN 6 //PWM frequency (default) 976.56 Hz
9 #define DIR_PIN 4
10 #define NOTEN_PIN 2
11
12 //We create an Elevator object
13 Elevator elevator(STEP_PIN, DIR_PIN, NOTEN_PIN);
14
15 //Order from the RPi that the elevator must follow
16 uint8_t order = 0; //Until the RPi sends an order for the first time, order=0 (no task to perform)
17
18 //FUNCTIONS USED FOR THE I2C COMMUNICATION
19
20 //Union used for the i2c communication
21 union ulongchar {
22     uint32_t l;
23     uint8_t b[4];
24 };
25
26 //Size of the message sent by the master, 1 byte (order to follow)
27 #define MESSAGE_LENGTH 1
28 //Size of the message requested by the master, 10 bytes ( 1B: direction, 1B: order,
29 //4B: position, 4B: target position)
30 #define REQUEST_MESSAGE_SIZE 10
31
32 //The message sent by the master is read
33 void receiveEvent(int howMany) {
34     //Protection to assure the number of bytes to read is equal to MESSAGE_LENGTH.
35     //If not, the slave doesn't read the message. As the master doesn't receive the acknowledge bit,
36     //it will resend it.
37     if (howMany != MESSAGE_LENGTH) {
38         while (Wire.available())
39             (uint8_t) Wire.read();
40         return;
41     }
42     order = (uint8_t) Wire.read(); //Size of the message sent by the master, 1 byte (order to follow)
43 }
44
45 void requestEvent() {
46     //The message requested by the master is sent
47     uint8_t exception = 0; //In this slave, no exceptions are thrown
48
49     uint8_t merge[REQUEST_MESSAGE_SIZE];
50     merge[0] = elevator.direction;
51     merge[1] = order;
52
53     ulongchar var;
54     var.l = elevator.height;
55     for (int i = 0; i < 4; i++)
56         merge[2 + i] = var.b[i];
57
58     var.l = elevator.heightTarget;
59
60     for (int i = 0; i < 4; i++)
61         merge[6 + i] = var.b[i];
62
63     Wire.write(merge, REQUEST_MESSAGE_SIZE);
64 }
65
66 void setup() {
67     //We initialize the i2c
68     Wire.begin(0x14);
69     Wire.onReceive(receiveEvent);
70     Wire.onRequest(requestEvent);
71 }
```

```
72
73
74 void loop() {
75     elevator.update();//At the beginning of each loop, and if the elevator is moving, its position is
    updated.
76     //If the target position is reached, stepper is stopped.
77
78     switch (order) { //We also check if the master sends an order (different from 0) and execute it.
79         case 1:
80             elevator.goTo(Elevator::BOTTOM);
81             order = 0;
82             break;
83         case 2:
84             elevator.goTo(Elevator::RISE_SEASHELL);
85             order = 0;
86             break;
87         case 3:
88             elevator.goTo(Elevator::HEIGHT48);
89             order = 0;
90             break;
91         case 4:
92             elevator.goTo(Elevator::HEIGHT64);
93             order = 0;
94             break;
95         case 5:
96             elevator.goTo(Elevator::PREPARATION_UMBRELLA);
97             order = 0;
98             break;
99         case 6:
100            elevator.goTo(Elevator::UMBRELLA);
101            order = 0;
102            break;
103        default: break;
104    }
105
106
107
108 }
109
```

```
1 #ifndef PINCE_ELEVATOR_H
2 #define PINCE_ELEVATOR_H
3
4 #include <Arduino.h>
5
6 /*
7  * class Elevator
8  *
9  * This class allows to control the movement of the elevator subsystem (containing the gripper and the
  umbrella)
10  * (by controlling the stepper motor SYS42STH38-1684A)
11  *
12  * All distances are in mm
13  */
14
15
16 #define DISTANCE_NB_STEPS 8/200
17 //Proportion: (distance covered with one revolution /number of steps par revolution)
18
19 class Elevator {
20 public:
21     //ENUMS
22     //Possible positions of the gripper
23     enum Position {
24         BOTTOM = 0, //The gripper is at its lowest position, right above the ground
25         RISE_SEASHELL = 30, //Position to rise a sheashell (to avoid dragging it or the gripper on the
  ground)
26         HEIGHT48 = 48, //Position to reach the seashells on the rocks, first level
27         HEIGHT64 = 64, //Position to reach the seashells on the rocks, second level
28         PREPARATION_UMBRELLA = 100, //Position to prepare the umbrella (before deploying it)
29         UMBRELLA = 130 //Umbrella completely deployed
30     };
31
32     //Directions of movement of the elevator
33     enum Direction {
34         UP,
35         DOWN
36     };
37
38     //METHODS
39
40     //Constructor
41     Elevator(uint8_t stepPin, uint8_t dirPin, uint8_t notEnablePin, Elevator::Position initialHeight =
  Elevator::BOTTOM);
42
43     void goTo(Elevator::Position position);
44     void update();
45     void stop();
46
47     //ATTRIBUTES
48     int32_t height; //Height of the gripper at each moment
49     int32_t heightStartMoving; //Height of the gripper before the displacements
50     int32_t heightTarget; //Height of the gripper to be reached at each moment
51     bool isMoving; //1:the elevator is moving, 2:the elevator isn't moving
52     Direction direction; //Direction of movement at each moment
53
54     unsigned long OverflowCountStartMoving; //Value of the timer0 overflow counter at the beginning of
  each
55     //new movement
56
57     //Pins used
58     uint8_t stepPin; //To control the taken steps
59     //We command steps by imposing pulses to this pin: analogWrite(stepPin,256/2)
60     //We stop the steps by stopping imposing pulses: analogWrite(stepPin,0)
61     //The speed (steps/sec) is the PWM frequency
62     uint8_t dirPin; //To control the direction
63     uint8_t notEnablePin; //To enable and disable the h-bridges connected to the motor coils
64
65 };
66
67 #endif //PINCE_ELEVATOR_H
```

```
1 #include "Elevator.h"
2
3
4 extern volatile unsigned long timer0_overflow_count;
5 /*External counter of the number of overflows of the Atmega's Timer0 since the beginning of the program.
6 *In this case, it will allow to count the number of steps undertaken by the stepper motor.
7 *
8 * Timer0 is an 8-bit timer fed by the 16 MHz clock divided by 64. Therefore, we can compute the number
9 * of overflows
10 * per unit of time as:
11 * (16*10^6/64 cycles/second)*(1/(2^8) overflows/cycle)= 976.5625 overflows/second
12 *
13 * Besides, the default PWM frequency is (16*10^6/(64*256))Hz= 976.5625 Hz. We use a PWM signal to
14 * command the
15 * steps of the stepper motor (imposing a rising edge makes the motor move one step). There's one step
16 * every PWM
17 * period, that is, 976.5625 step/second.
18 *
19 * As the number of overflows per second and the number of steps per second is the same,
20 * if we count the number of overflows between two instants of time when the stepper motor is running,
21 * we'll be also
22 * counting the number of steps between those instants.
23 * */
24
25 /*
26 * Constructor
27 */
28 Elevator::Elevator(uint8_t stepPin, uint8_t dirPin, uint8_t notEnablePin, Elevator::Position
29 initialHeight, ) :
30     stepPin(stepPin), dirPin(dirPin), notEnablePin(notEnablePin), height(initialHeight) {
31     //We define the pin modes
32     pinMode(stepPin, OUTPUT);
33     pinMode(dirPin, OUTPUT);
34     pinMode(notEnablePin, OUTPUT);
35
36     //Initially
37     digitalWrite(notEnablePin, HIGH); //H-bridges are disabled
38     isMoving = false;
39     heightTarget = initialHeight;
40 }
41
42 /*
43 * update()
44 * This method is called at the beginning of each loop of the main program.
45 *
46 * Enables or disables the H-bridges depending on whether the elevator is supposed to move or not (value
47 * of isMoving)
48 * Updates the position of the elevator (height)
49 * and stops it if it has reached the target value (heightTarget)
50 */
51 void Elevator::update() {
52     //if isMoving==0, we disable the h-bridges
53     if (!isMoving) {
54         digitalWrite(notEnablePin, HIGH);
55         return;
56     }
57     //if isMoving==1, we enable the h-bridges
58     digitalWrite(notEnablePin, LOW);
59
60     if (direction == UP) {
61         //We update the position
62         this->height = (int32_t) (heightStartMoving +
63             (timer0_overflow_count - OverflowCountStartMoving) *
64             DISTANCE_NB_STEPS);
65
66         if (this->height > heightTarget) { //Target reached, the elevator is stopped
67             isMoving = false;
68             stop();
69         }
70     }
71 }
```

```
65     }
66     else{
67         //We update the position
68         this->height = (int32_t) (heightStartMoving -
69                                 (timer0_overflow_count - OverflowCountStartMoving) *
        DISTANCE_NB_STEPS);
70         if (this->height < heightTarget) { //Target reached, the elevator is stopped
71             isMoving = false;
72             stop();
73         }
74     }
75
76 }
77 /*
78  * stop()
79  * Method called by other methods of Elevator.
80  *
81  * It stops the elevator
82  */
83 void Elevator::stop() {
84     analogWrite(stepPin, 0);
85 }
86
87 /*
88  * goTo(Elevator::Position position)
89  * Method called on the loop of the main program, each time the master sends an order.
90  *
91  * With the position ordered by the master (position) and the actual position of the gripper (height),
92  * it deduces if the elevator should go up or down and starts the movement
93  */
94 void Elevator::goTo(Elevator::Position position) {
95     heightTarget = position;
96     heightStartMoving = height;
97     OverflowCountStartMoving = timer0_overflow_count;
98
99     if (this->height > heightTarget) { //The gripper has to go down
100         direction = DOWN;
101         digitalWrite(dirPin, HIGH);
102
103         isMoving = true;
104         analogWrite(stepPin, 256 / 2);
105     }
106     else if (this->height < heightTarget) { //The gripper has to go up
107         direction = UP;
108         digitalWrite(dirPin, LOW);
109         isMoving = true;
110         analogWrite(stepPin, 256 / 2);
111     }
112     else { //The gripper doesn't have to move
113         analogWrite(stepPin, 0);
114     }
115 }
116
```

2.2. SLAVE GRIPPER

```
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include "pwm.h"
4 #include "Gripper.h"
5
6
7
8
9
10 //PIN VALUES
11 #define ENABLE_PIN 8
12 #define INPUT1_PIN 9
13 #define INPUT2_PIN 10
14 #define VREF_PIN A2
15 #define VOUT_PIN A1
16 #define POTENTIOMETRE_PIN A0
17
18 //We create a gripper object
19 Gripper gripper(ENABLE_PIN, INPUT1_PIN, INPUT2_PIN, VREF_PIN, VOUT_PIN, POTENTIOMETRE_PIN);
20
21 //Order from the RPi that the gripper must follow
22 uint8_t order = 0; //Until the RPi sends an order for the first time, order=0 (no task to perform)
23
24 //FUNCTIONS USED FOR THE I2C COMMUNICATION
25
26 //Unions used for the i2c communication
27 union ulongchar {
28     uint32_t l;
29     uint8_t b[4];
30 };
31
32 union usmallchar {
33     uint16_t s;
34     uint8_t b[2];
35 };
36
37 union floatchar {
38     float f;
39     uint8_t b[4];
40 };
41
42 //Size of the message sent by the master, 1 byte (order to follow)
43 #define MESSAGE_LENGTH 1
44 //Size of the message requested by the master, 12 bytes (1B: exception,
45 // 1B: order to perform, 1B: task being performed,
46 // 1B: movement, 1B: error, 1B: isGrabbing, 2B: angle, 4B: intensity)
47 #define REQUEST_MESSAGE_SIZE 12
48
49 // function call when the master send a message
50 void receiveEvent(int howMany) {
51     //Protection to assure the number of bytes to read is equal to MESSAGE_LENGTH.
52     //If not, the slave doesn't read the message. As the master doesn't receive the acknowledge bit,
53     //it will resend it.
54     if (howMany != MESSAGE_LENGTH) {
55         while (Wire.available())
56             (uint8_t) Wire.read();
57         return;
58     }
59     order = (uint8_t) Wire.read();
60 }
61
62
63
64 //The message requested by the master is sent
65 void requestEvent() {
66     uint8_t exception = 0; //In this slave, no exceptions are thrown
67
68     uint8_t merge[REQUEST_MESSAGE_SIZE];
69     merge[0] = exception;
70     merge[1] = order;
71     merge[2] = (uint8_t) gripper.task;
```

```
72  merge[3] = (uint8_t) gripper.movement;
73  merge[4] = (uint8_t) gripper.error;
74  merge[5] = (uint8_t) gripper.isGrabbing;
75
76  usmallchar var;
77  var.s = gripper.potentiometer->getValue();
78  for(int i=0; i<2; i++){
79      merge[6+i] = var.b[i];
80  }
81
82  floatchar floatchar1;
83  floatchar1.f = gripper.servomotor->readIntensity();
84  for(int i=0; i<4; i++){
85      merge[8+i] = floatchar1.b[i];
86  }
87
88  Wire.write(merge, REQUEST_MESSAGE_SIZE);
89 }
90
91 //////////////////////////////////////////////////
92 // Core of the code //
93 //////////////////////////////////////////////////
94
95 void setup() {
96     // We initialize the i2c
97     Wire.begin(I2C_ADDR);
98     Wire.onReceive(receiveEvent);
99     Wire.onRequest(requestEvent);
100
101     // Modification of the PWM
102     changeClockDivider(1,1);
103
104     //We initialize the gripper
105     gripper.init();
106
107 }
108
109 void loop() {
110     gripper.updateReal(); //Attributes movement and task are updated
111
112     if(order != 0){
113         // followOrder is call if the master sends an order different from 0
114         gripper.followOrder(order);
115         order = 0;
116     }
117 }
118
119
120
121
```

```
1 #ifndef SLAVE_PINCE_CONSTANTES_H
2 #define SLAVE_PINCE_CONSTANTES_H
3
4
5 /*
6  * SLAVE GRIPPER CONSTANTS
7  * NOTE: the ANGLES-, the INTENSITIES and the DUTY CYCLES have to be recalibrated
8  * if we change the gripper mechanics, with empiric tests.
9  */
10
11 //i2c address of this slave
12 #define I2C_ADDR 0x16
13
14 //ANGLES (= measure of the potentiometer voltage)
15
16 // If the value of the potentiometer is greater or lower than these limits,
17 // the gripper movement has to be stopped
18 #define ANGLE_LIMIT_OPEN (ANGLE_ERROR_CLOSE+110)
19 #define ANGLE_LIMIT_CLOSE (ANGLE_ERROR_CLOSE+30)
20
21 // If the value of the potentiometer is greater or lower than these limits,
22 // this is not normal. ERROR.
23 // The gripper movement has to be stopped.
24 #define ANGLE_ERROR_OPEN (ANGLE_ERROR_CLOSE+320-150)
25 #define ANGLE_ERROR_CLOSE 130
26
27
28
29 //Absurd angle. If the measured angle is greater than this upper-bound limit,
30 // we know the measure is incorrect.
31 #define ANGLE_ABSURD 400
32
33 //INTENSITIES
34 //The measured intensity should not go over this value. ERROR.
35 #define INTENSITY_ERROR 2.00
36
37 //If the measured intensity is greater than this limit,
38 // we consider that the gripper is grabbing an object or blocked by an object
39 #define INTENSITY_THRESHOLD 1.00
40
41 //INTENSITY MEASURE CIRCUIT PARAMETERS
42 #define GAIN_INST_AMPLIFIER 101
43 #define KELVIN_RESISTOR_VALUE 0.01 //ohms
44
45 /*
46  * DUTY CYCLES (with which the motor is commanded)
47  *
48  * In our circuit: alpha>0 - opening, alpha<0 - closing
49  */
50
51 #define ALPHA_OPENING 170 //To open a gripper with no load
52 #define ALPHA_CLOSING -170//To close a gripper with no load
53 #define ALPHA_SEASHELL 255//To grab and hold a seashell
54
55
56
57 #endif //SLAVE_PINCE_CONSTANTES_H
58
```

```
1 #ifndef DEF_PWM
2 #define DEF_PWM
3
4 #include <Arduino.h>
5
6 /**
7  * FUNCTION COURTESY OF THE SUB-TEAM MOTEURS (CROC2016)
8  * This function allows to change the PWM frequency
9  *
10 * Usage:
11 * - counterId: allows to choose the counter (look at the mapping below)
12 * - divider: allows to choose the divider (look at the possible values below)
13 * To have a PWM frequency of f Hz, divider = 16e6 / (512 * f)
14 * explicación se mantiene??
15 *
16 * The Atmega328p has three counters that allow to command the rising and falling edges
17 * on the PWM pins. http://www.atmel.com/Images/doc8161.pdf section 17.2
18 *
19 * The input of these counters is a clock signal modified with prescalers.
20 * These divide the frequency of the microcontroller's main clock by a factor of:
21 * 1, 8, 32, 64, 128, 256 or 1024. Look at table 17.9.
22 *
23 * The choice of this factor is made by changing the last three bits (CSn0, CSn1 and CSn2)
24 * of the TCCRnB register (n is the counter index).
25 *
26 * Mapping: Arduino Uno pin => counter
27 * https://www.arduino.cc/en/Hacking/PinMapping168
28 * 5 => 0
29 * 6 => 0
30 * 9 => 1
31 * 10 => 1
32 * 3 => 2
33 * 11 => 2
34 */
35
36 void changeClockDivider(int counterId, int divider);
37
38 #endif
39
```

```
1 #include "pwm.h"
2
3 void changeClockDivider(int counterId, int divider) {
4     bool bit0 = divider == 1 || divider == 32 || divider == 128 || divider == 1024;
5     bool bit1 = divider == 8 || divider == 32 || divider == 256 || divider == 1024;
6     bool bit2 = divider >= 64;
7     volatile uint8_t *reg = NULL;
8     switch (counterId) {
9         case 0:
10             reg = &TCCR0B;
11             break;
12         case 1:
13             reg = &TCCR1B;
14             break;
15         case 2:
16             reg = &TCCR2B;
17             break;
18         default:
19             break;
20     }
21     if (reg != NULL) {
22         (*reg) |= bit0 * _BV(CS20);
23         (*reg) |= bit1 * _BV(CS21);
24         (*reg) |= bit2 * _BV(CS22);
25         (*reg) &= ~((!bit0) * _BV(CS20));
26         (*reg) &= ~((!bit1) * _BV(CS21));
27         (*reg) &= ~((!bit2) * _BV(CS22));
28     }
29 }
30 }
```

```
1 #ifndef SLAVE_PINCE_POTENTIOMETRE_H
2 #define SLAVE_PINCE_POTENTIOMETRE_H
3
4 #include <Arduino.h>
5 /*
6  * class Potentiometer
7  *
8  * This class allows to measure the voltage between the terminals of the servo motor potentiometer (
  therefore,
9  * its rotation angle)
10 *
11 * We directly work with the values given by analogRead, no conversion is done.
12 */
13 class Potentiometer {
14 public:
15     //METHODS
16     Potentiometer(uint8_t pin);
17     void init();
18     uint16_t getValue();
19
20     //ATTRIBUTES
21     uint8_t measurePin; //Microcontroller pin used to measure the potentiometer voltage
22     uint16_t previousValue; //Last value of the angle measure
23
24 };
25
26
27 #endif //SLAVE_PINCE_POTENTIOMETRE_H
28
```

```
1 #include "Potentiometer.h"
2 #include "constantes.h"
3 /*
4  * Constructor
5  */
6 Potentiometer::Potentiometer(uint8_t pin) : measurePin(pin), previousValue{0} {
7
8 }
9
10 /*
11  * init()
12  * This method initializes the measure pin
13  */
14 void Potentiometer::init () {
15     pinMode(this->measurePin, INPUT);
16 }
17
18 /*
19  * getValue()
20  * This method allows to read the measure and update previousValue
21  */
22 uint16_t Potentiometer::getValue () {
23     int value = analogRead(this->measurePin);
24     if (value < ANGLE_ABSURD) //PROTECTION: the measure is greater than ANGLE_ABSURD we know that it's
        not correct.
25         previousValue = (uint16_t) value;
26     return previousValue;
27 }
28
29 /*
30  * NOTE: We decided to create ANGLE_ABSURD since we sometimes obtained absurd measure values
31  * and we weren't able to determine their origin. We determined, empirically, that all absurd
32  * values were greater than ANGLE_ABSURD and we could establish this limit to detect them.
33  * Later, we could figure out the origin of these errors (damaged wire). The problem was solved
34  * but we decided to not remove the protection
35  */
36
37
38
39
```

```
1 #ifndef INC_20160418_SERVOMOTOR_H
2 #define INC_20160418_SERVOMOTOR_H
3
4
5 #include <Arduino.h>
6 /*
7  * class Servomotor
8  *
9  * This class allows to control the gripper servo motor (with the new electronic circuit).
10  * It allows to command its rotation and measure its intensity.
11  */
12 class Servomotor {
13 public:
14     //METHODS
15     Servomotor(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t pinVout
16 );
17     float readIntensity();
18     void writeAlphaCommand(int16_t alpha);
19     void init();
20
21     //ATTRIBUTES
22     uint8_t pinEnable; //Pin used to enable the H-bridge
23     uint8_t pinInput1; //Pin used to control the H-bridge input1
24     uint8_t pinInput2; //Pin used to control the H-bridge input2
25     uint8_t pinVref; //Pin used to measure Vref (voltage reference of the inst. amplifier)
26     uint8_t pinVout; //Pin used to measure Vout (output voltage of the inst. amplifier)
27 };
28
29
30 #endif //INC_20160418_REGLAGEPIDSERVOMOTOR_SERVOMOTOR_H
31
```

```
1 #include "Servomotor.h"
2 #include "constantes.h"
3 /*
4  * Constructor
5  */
6 Servomotor::Servomotor(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t
  pinVout) :
7     pinEnable(pinEnable), pinInput1(pinInput1), pinInput2(pinInput2), pinVref(pinVref), pinVout(
  pinVout){}
8
9 /*
10 * readIntensity()
11 */
12
13 float Servomotor::readIntensity() {
14
15     //We compute the average of the measures of (Vout-Vref) during 10ms,
16     // to avoid spikes that may result false measures.
17     //Hypothesis: Vout-Vref sign doesn't change within 10 ms
18     float moyenne = abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
19     delay(1);
20     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
21     delay(1);
22     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
23     delay(1);
24     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
25     delay(1);
26     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
27     delay(1);
28     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
29     delay(1);
30     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
31     delay(1);
32     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
33     delay(1);
34     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
35     delay(1);
36     moyenne += abs((analogRead(this->pinVout) - analogRead(this->pinVref)));
37
38     //We compute the intensity corresponding to (Vout-Vref) measured.
39     float vAmplified = moyenne * 5 / 1024.0 / 10;
40     float intensity = vAmplified / (GAIN_INST_AMPLIFIER * KELVIN_RESISTOR_VALUE);
41     return intensity;
42 }
43 /*
44 * writeAlphaCommand()
45 * This method allows to command the rotation of the motor. If alphaCommand( duty cycle)>0, it will
  rotate
46 * in one direction. If alpha>0 in the other. If alpha=0, it's stopped.
47 */
48
49 void Servomotor::writeAlphaCommand(int alphaCommand) {
50     // We assure -255<= alphaCommand <=255
51     if (alphaCommand > 255) {
52         alphaCommand = 255;
53     } else if (alphaCommand < -225) {
54         alphaCommand = -255;
55     }
56
57
58     if (alphaCommand >= 0) {
59         digitalWrite(pinEnable, HIGH);
60         digitalWrite(pinInput2, LOW);
61         analogWrite(pinInput1, alphaCommand);
62     } else {
63         digitalWrite(pinEnable, HIGH);
64         digitalWrite(pinInput1, LOW);
65         analogWrite(pinInput2, -alphaCommand);
66     }
67 }
68
```

```
69 /*
70  * init()
71  * This method initializes the pins
72  */
73 void Servomotor::init() {
74     pinMode(pinEnable, OUTPUT);
75     pinMode(pinInput1, OUTPUT);
76     pinMode(pinInput2, OUTPUT);
77     pinMode(pinVout, INPUT);
78     pinMode(pinVref, INPUT);
79     digitalWrite(pinEnable, LOW);
80     digitalWrite(pinInput1, LOW);
81     digitalWrite(pinInput2, LOW);
82 }
83
```

```
1 #ifndef SLAVE_PINCE_PINCE_H
2 #define SLAVE_PINCE_PINCE_H
3
4 #include <Arduino.h>
5 #include "Servomotor.h"
6 #include "Potentiometer.h"
7 #include "constantes.h"
8
9 //ENUMS
10
11 //Task the gripper must do (following an order sent by the master)
12 enum Task {
13     NONE,
14     OPEN_COMPLETELY,
15     CLOSE_COMPLETELY,
16     GRAB_SEASHELL,
17 };
18
19
20 //Action or movement the gripper can perform
21 enum Movement {
22     STOPPED,
23     OPENING,
24     CLOSING,
25     GRABBING_SEASHELL,
26     ERROR
27 };
28
29 //Possible errors of the system
30 enum Error {
31     NONE_ERR,
32     ERR_TOO_OPEN,
33     ERR_TOO_CLOSE,
34     ERR_TOO_MUCH_INT,
35     ERR_OBSTRUCTION_OPENING,
36     ERR_OBSTRUCTION_CLOSING,
37 };
38 };
39 /*
40 * class Gripper
41 *
42 * This class allows to control the whole gripper system.
43 */
44 class Gripper {
45 public:
46
47     //METHODS
48     Gripper(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t pinVout,
49     uint8_t pinPotentiometer);
50     void init();
51     void followOrder(uint8_t order);
52     void updateReal();
53     void openGripper();
54     void closeGripper();
55     void stop();
56     void grabSomething(uint8_t alpha);
57
58     //ATTRIBUTES
59     //The gripper includes a servomotor and a potentiometer
60     Servomotor *servomotor;
61     //For our mechanical assembly, if we command sevomotor->writeAlphaCommand(int alphaCommand)
62     // with alphaCommand<0 the griper CLOSES. With alphaCommand>0 it OPENS.
63     Potentiometer *potentiometer;
64
65     float intensity; //Intensity flowing through the gripper's servomotor
66     int angle; //Angle of the gripper
67
68     Movement movement; //Movement of the gripper performs at each moment
69     Task task; //Task the gripper follows at each moment
70     bool isGrabbing; //1: the gripper is loaded with an object
71     Error error; //Error at each moment
```

```
71 }  
72  
73 #endif //SLAVE_PINCE_PINCE_H  
74
```

```
1 #include "Gripper.h"
2
3
4 /*
5  * Constructor
6  */
7 Gripper::Gripper(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t
  pinVout,
8                  uint8_t pinPotentiometer) {
9     potentiometer = new Potentiometer(pinPotentiometer);
10    servomotor = new Servomotor(pinEnable, pinInput1, pinInput2, pinVref, pinVout);
11 }
12
13 /*
14  * followOrder()
15  *
16  * This method is called in the loop of the main program,
17  * each time the master sends an order different from 0.
18  * According to the sent variable, the function updates the attributes task and movement.
19  */
20 void Gripper::followOrder(uint8_t order) {
21     switch (order) {
22         case OPEN_COMPLETELY:
23             this->movement = OPENING;
24             this->task = OPEN_COMPLETELY;
25
26             break;
27         case CLOSE_COMPLETELY:
28             this->movement = CLOSING;
29             this->task = CLOSE_COMPLETELY;
30             break;
31         case GRAB_SEASHELL:
32             this->movement = GRABBING_SEASHELL;
33             this->task = GRAB_SEASHELL;
34             break;
35         case NONE:
36             break;
37         default:
38             break;
39     }
40 }
41
42
43 /*
44  * updateReal()
45  *
46  * This method is called at the beginning of each loop of the main program.
47  *
48  * It allows to command different actions (e.g. calling methods as closeGripper()) depending on the
49  * value of movement. It also updates the values of intensity and angle and detects intensity and angle
50  * errors.
51  */
52 void Gripper::updateReal() {
53     //If the movement is other than ERROR, then there's no error in the system
54     if (movement != ERROR)
55         error = NONE_ERR;
56
57     //We update the values of angle and intensity
58     this->angle = potentiometer->getValue();
59     this->intensity = servomotor->readIntensity();
60
61     //We assure these values haven't reached the error limits
62     if (this->angle > ANGLE_ERROR_OPEN) {
63         stop();
64         this->error = ERR_TOO_OPEN;
65         this->movement = ERROR;
66         return;
67     } else if (this->angle < ANGLE_ERROR_CLOSE) {
68         stop();
69         this->error = ERR_TOO_CLOSE;
70         this->movement = ERROR;
```

```
71     return;
72 } else if (this->intensity > INTENSITY_ERROR) {
73     stop();
74     this->error = ERR_TOO_MUCH_INT;
75     this->movement = ERROR;
76     return;
77 }
78
79 //Depending on the value of the attribute movement, an action is commanded
80 switch (movement) {
81     case OPENING:
82         openGripper();
83         break;
84     case CLOSING:
85         closeGripper();
86         break;
87     case STOPPED:
88         stop();
89         break;
90     case GRABBING_SEASHELL:
91         grabSomething(ALPHA_SEASHELL);
92         break;
93     case ERROR:
94         this->stop();
95         break;
96 }
97 }
98
99 /*
100 * openGripper()
101 *
102 * This method is called to open the gripper.
103 */
104 void Gripper::openGripper() {
105     //If angle>=ANGLE_LIMIT_OPEN the gripper has finished opening, it stops.
106     if (this->angle >= ANGLE_LIMIT_OPEN) {
107         this->stop();
108         this->movement = STOPPED;
109         return;
110     }
111
112
113     //If intensity > INTENSITY_THRESHOLD there's an object blocking the opening
114     if (this->intensity > INTENSITY_THRESHOLD) {
115         this->error = ERR_OBSTRUCTION_OPENING;
116         this->movement = ERROR;
117         return;
118     }
119
120     //Else, the gripper continues opening
121     this->servomotor->writeAlphaCommand(ALPHA_OPENING);
122     this->isGrabbing = false; //We can assure the gripper isn't loaded
123 }
124
125 /*
126 * closeGripper()
127 *
128 * This method is called to close the gripper, without grabbing an object
129 */
130 void Gripper::closeGripper() {
131     //If angle<=ANGLE_LIMIT_CLOSE the gripper has finished closing, it stops.
132     if (this->angle <= ANGLE_LIMIT_CLOSE) { //Gripper a fini de se fermer sans rien saisir
133         this->movement = STOPPED;
134         return;
135     }
136
137     //If intensity > INTENSITY_THRESHOLD there's an object blocking the closing
138     if (this->intensity >= INTENSITY_THRESHOLD) {
139
140         this->error = ERR_OBSTRUCTION_CLOSING;
141         this->movement = ERROR;
```

```
142         return;
143     }
144
145
146     //Else, the gripper continues closing
147     this->servomotor->writeAlphaCommand(ALPHA_CLOSING);
148     this->isGrabbing = false; //We can assure the gripper isn't loaded
149 }
150
151 /*
152  * stop()
153  *
154  * This method stops the servomotor. After doing so, attribute task is updated to NONE
155  */
156 void Gripper::stop() {
157     this->servomotor->writeAlphaCommand(0);
158     this->task = NONE;
159 }
160
161 /*
162  * grabSomething(uint8_t alpha)
163  *
164  * This method allows to grab an object, specifying with which duty cycle the motor rotation
165  * has to be commanded.
166  * When the gripper grabs an object (motor blocked) the duty cycle is related to the torque
167  * applied by the motor (the greater duty cycle, the greater torque)
168  */
169
170 void Gripper::grabSomething(uint8_t alpha) {
171     //alpha has to be positif, in this case it's known that the gripper has to
172     // close and it's later transformed into negative
173
174     //If angle<=ANGLE_LIMIT_CLOSE the gripper has finished closing, it stops.
175     if (this->angle <= ANGLE_LIMIT_CLOSE) {
176         this->movement = STOPPED;
177         return;
178     }
179
180     //Else, the gripper "tries" to grab the object
181     this->servomotor->writeAlphaCommand(-alpha);
182     // If intensity > INTENSITY_THRESHOLD an object is gripped
183     this->isGrabbing = this->intensity> INTENSITY_THRESHOLD;
184 }
185
186 /*
187  * init()
188  *
189  * This method initializes the attributes of the class, the potentiometer and servomotor objects
190  */
191 void Gripper::init() {
192     this->angle = potentiometer->getValue();
193     this->intensity = servomotor->readIntensity();
194     this->error = NONE_ERR;
195     this->movement = STOPPED;
196     this->task = NONE;
197     this->isGrabbing = false;
198     this->potentiometer->init();
199     this->servomotor->init();
200 }
201
202
203
204
205
206
207
208
209
210
```

2.3. TEST CLASS COLOR

```
1 #include "Arduino.h"
2 #include "Color.h"
3
4 //Arduino Uno pins used
5 #define S0 8
6 #define S1 9
7 #define S2 10
8 #define S3 11
9 #define S4 12
10
11 //Colors stored in memory
12 Color violet(235,117,112,425);
13
14 Color vert(175,113,174,420);
15
16 Color blanc(747,359,506,1563);
17
18 Color orange(434,121,212,711);
19
20 unsigned long timeBef; //Variable to compute duration of the color detection
21
22 //Variables to store the measured frequencies
23 float fClear; //No filter
24 float fRed; //Red filter
25 float fGreen; //Green filter
26 float fBlue; //Blue filter
27
28 //Variables to store the distance to the colors in memory
29 double dViolet;
30 double dVert;
31 double dOrange;
32 double dBlanc;
33
34 //Function to obtain a frequency in Hz from a half-period in microseconds
35 float frequency(unsigned long halfPeriod);
36
37 void setup() {
38     //TCS3200 setup
39     pinMode(S0, OUTPUT);
40     pinMode(S1, OUTPUT);
41     pinMode(S2, OUTPUT);
42     pinMode(S3, OUTPUT);
43     pinMode(S4, INPUT);
44     digitalWrite(S0, LOW);
45     digitalWrite(S1, HIGH);
46     //Serial communication setup
47     Serial.begin(9600);
48 }
49
50
51 void loop() {
52     //We measure the frequencies
53     Serial.println("MEASURE STARTS");
54     timeBef=micros();
55     //NO FILTER
56     digitalWrite(S2, HIGH);
57     digitalWrite(S3, LOW);
58     fClear=frequency(pulseIn(S4, HIGH));
59
60     //RED FILTER
61     digitalWrite(S2, LOW);
62     digitalWrite(S3, LOW);
63     fRed=frequency(pulseIn(S4, HIGH));
64
65     //BLUE FILTER
66     digitalWrite(S2, LOW);
67     digitalWrite(S3, HIGH);
68     fBlue=frequency(pulseIn(S4, HIGH));
69
70     //GREEN FILTER
71     digitalWrite(S2, HIGH);
```

```
72 digitalWrite(S3,HIGH);
73 fGreen=frequency(pulseIn(S4,HIGH));
74
75
76 //We compute the measures
77 dViolet=violet.distance(fRed,fBlue,fGreen,fClear);
78 dVert=vert.distance(fRed,fBlue,fGreen,fClear);
79 dOrange=orange.distance(fRed,fBlue,fGreen,fClear);
80 dBlanc=blanc.distance(fRed,fBlue,fGreen,fClear);
81
82 //We deduce the color (corresponding to the smallest distance)
83 if(dViolet<=dVert && dViolet<=dOrange && dViolet<=dBlanc){
84     Serial.println("Violet");
85 }else if(dVert<=dViolet && dVert<=dOrange && dVert<=dBlanc){
86     Serial.println("Vert");
87 }else if (dOrange<=dViolet && dOrange<=dVert && dOrange<=dBlanc){
88     Serial.println("Orange");
89 }else{
90     Serial.println("Blanc");
91 }
92
93 Serial.println("Measure duration in microseconds: ");
94 Serial.print(micros()-timeBef);
95 Serial.println("5 before the next measure");
96 delay(5000);
97
98 }
99
100 float frequency(unsigned long halfPeriod){
101
102     return (float) (500000.0/(halfPeriod));
103 }
104
105
```

```
1 //
2 // Created by CANDELA on 13/06/2016.
3 //
4
5 #ifndef LOGIQUE1306_COLOR_H
6 #define LOGIQUE1306_COLOR_H
7
8
9 class Color {
10
11 public:
12
13     float redN; //All colors are normalized with respect to the sum of the three values
14     float blueN;
15     float greenN;
16     Color(float red, float blue, float green, float clr);
17     double distance(float red, float blue, float green, float clr);
18
19
20 };
21
22
23 #endif //LOGIQUE1306_COLOR_H
24
```

```
1 //
2 // Created by CANDELA on 13/06/2016.
3 //
4
5 #include "Color.h"
6 #include "Arduino.h"
7
8 Color::Color(float red, float blue, float green, float clr) {
9     this->redN=red/clr*100;
10    this->blueN=blue/clr*100;
11    this->greenN=green/clr*100;
12 }
13
14 double Color::distance(float red, float blue, float green, float clr) {
15
16     red=red/clr*100;
17     blue=blue/clr*100;
18     green=green/clr*100;
19
20     return sqrt(pow(red-this->redN,2)+pow(blue-this->blueN,2)+pow(green-this->greenN,2));
21
22 }
23
```

2.4. SLAVE GRIPPER EXTENDED (COLOR SENSOR)

```
1 #ifndef SLAVE_PINCE_CONSTANTES_H
2 #define SLAVE_PINCE_CONSTANTES_H
3
4
5 /*
6  * SLAVE GRIPPER EXTENDED CONSTANTS
7  * NOTE: the ANGLES-, the INTENSITIES and the DUTY CYCLES have to be recalibrated if we change the
8  * gripper mechanics,
9  * with empiric tests.
10 */
11 //i2c address of this slave
12 #define I2C_ADDR 0x16
13
14 //ANGLES (= measure of the potentiometer voltage)
15
16 // If the value of the potentiometer is greater or lower than these limits, the gripper movement has to
17 // be stopped
18 #define ANGLE_LIMIT_OPEN (ANGLE_ERROR_CLOSE+110)
19 #define ANGLE_LIMIT_CLOSE (ANGLE_ERROR_CLOSE+30)
20
21 // If the value of the potentiometer is greater or lower than these limits, this is not normal. ERROR.
22 // The gripper movement has to be stopped.
23 #define ANGLE_ERROR_OPEN (ANGLE_ERROR_CLOSE+320-150)
24 #define ANGLE_ERROR_CLOSE 130
25
26 //Absurd angle. If the measured angle is greater than this upper-bound limit, we know the measure is
27 // incorrect.
28 #define ANGLE_ABSURD 400
29
30 //INTENSITIES
31 //The measured intensity should not go over this value. ERROR.
32 #define INTENSITY_ERROR 2.00
33
34 //If the measured intensity is greater than this limit, we consider that the gripper is grabbing an
35 // object or blocked
36 //by an object
37 #define INTENSITY_THRESHOLD 1.00
38
39 //INTENSITY MEASURE CIRCUIT PARAMETERS
40 #define GAIN_INST_AMPLIFIER 101
41 #define KELVIN_RESISTOR_VALUE 0.01 //ohms
42
43 /*
44  * DUTY CYCLES (with which the motor is commanded)
45  *
46  * In our circuit: alpha>0 - opening, alpha<0 - closing
47  */
48
49 #define ALPHA_OPENING 170 //To open a gripper with no load
50 #define ALPHA_CLOSING -170//To close a gripper with no load
51 #define ALPHA_SEASHELL 255//To grab and hold a seashell
52
53 //Timeout for pulseIn() in microseconds, used in Taos::readFrequency()
54 #define TIMEOUT_PULSEIN 50000
55
56 #endif //SLAVE_PINCE_CONSTANTES_H
```

```
1 #include <Arduino.h>
2 #include <Wire.h>
3 #include "pwm.h"
4 #include "Gripper.h"
5
6
7
8
9
10 //PIN VALUES
11 #define ENABLE_PIN 8
12 #define INPUT1_PIN 9
13 #define INPUT2_PIN 10
14 #define VREF_PIN A2
15 #define VOUT_PIN A1
16 #define POTENTIOMETER_PIN A0
17 #define S0_PIN 2
18 #define S1_PIN 3
19 #define S2_PIN 4
20 #define S3_PIN 5
21 #define OUT_PIN 6
22
23 //We create a gripper object
24 Color purple(235,117,112,425,0);
25 Color green(175,113,174,420,1);
26 Color white(747,359,506,1563,2);
27 Color orange(434,121,212,711,3);
28 Color possibleColors[4]={purple,green,white,orange};
29 Gripper gripper(ENABLE_PIN, INPUT1_PIN, INPUT2_PIN, VREF_PIN, VOUT_PIN, POTENTIOMETER_PIN,S0_PIN,S1_PIN,
  S2_PIN,S3_PIN,OUT_PIN,4,possibleColors);
30
31 //Order from the RPi that the gripper must follow
32 uint8_t order = 0; //Until the RPi sends an order for the first time, order=0 (no task to perform)
33
34 //FUNCTIONS USED FOR THE I2C COMMUNICATION
35
36 //Unions used for the i2c communication
37 union ulongchar {
38     uint32_t l;
39     uint8_t b[4];
40 };
41
42 union usmallchar {
43     uint16_t s;
44     uint8_t b[2];
45 };
46
47 union floatchar {
48     float f;
49     uint8_t b[4];
50 };
51
52 //Size of the message sent by the master, 1 byte (order to follow)
53 #define MESSAGE_LENGTH 1
54 //Size of the message requested by the master, 12 bytes (1B: exception,
55 // 1B:order to perform,1B: task being performed,
56 // 1B: movement, 1B: error, 1B:isGrabbing, 2B: angle, 4B:intensity,1B: lastReadColor)
57 #define REQUEST_MESSAGE_SIZE 13
58
59 // function call when the master send a message
60 void receiveEvent(int howMany){
61     //Protection to assure the number of bytes to read is equal to MESSAGE_LENGTH.
62     //If not, the slave doesn't read the message. As the master doesn't receive the acknowledge bit,
63     //it will resend it.
64     if (howMany != MESSAGE_LENGTH){
65         while (Wire.available())
66             (uint8_t) Wire.read();
67         return;
68     }
69     order = (uint8_t) Wire.read();
70 }
```

```
71
72
73
74 //The message requested by the master is sent
75 void requestEvent() {
76     uint8_t exception = 0; //In this slave, no exceptions are thrown
77
78     uint8_t merge[REQUEST_MESSAGE_SIZE];
79     merge[0] = exception;
80     merge[1] = order;
81     merge[2] = (uint8_t) gripper.task;
82     merge[3] = (uint8_t) gripper.movement;
83     merge[4] = (uint8_t) gripper.error;
84     merge[5] = (uint8_t) gripper.isGrabbing;
85
86     usmallchar var;
87     var.s = gripper.potentiometer->getValue();
88     for(int i=0; i<2; i++){
89         merge[6+i] = var.b[i];
90     }
91
92     floatchar floatchar1;
93     floatchar1.f = gripper.servomotor->readIntensity();
94     for(int i=0; i<4; i++){
95         merge[8+i] = floatchar1.b[i];
96     }
97
98     merge[12]=grripper.lastReadColor;
99     Wire.write(merge, REQUEST_MESSAGE_SIZE);
100 }
101
102 ////////////////
103 // Core of the code //
104 ////////////////
105
106 void setup() {
107     // We initialize the i2c
108     Wire.begin(I2C_ADDR);
109     Wire.onReceive(receiveEvent);
110     Wire.onRequest(requestEvent);
111
112     // Modification of the PWM
113     changeClockDivider(1,1);
114
115     //We initialize the gripper
116     gripper.init();
117
118 }
119
120 void loop() {
121     gripper.updateReal(); //Attributes movement and task are updated
122
123     if(order != 0){
124         // followOrder is call if the master sends an order different from 0
125         gripper.followOrder(order);
126         order = 0;
127     }
128 }
129
130
131
132
```

```
1
2 #ifndef PROJECT_COLOR_H
3 #define PROJECT_COLOR_H
4 #include <Arduino.h>
5 /*
6  * class Color
7  *
8  * This class represents a vector of a color system. Each color has three components and an index.
9  * The method distance allows to compute the Euclidian distance to the vector.
10 */
11 class Color {
12 public:
13
14     //METHODS
15     Color(float red, float blue, float green, float clr, uint8_t index);
16     double distance(float red, float blue, float green, float clr);
17     //ATTRIBUTES
18     uint8_t index;
19     //Components of the vector
20     float redN;
21     float blueN;
22     float greenN;
23
24
25 };
26
27
28 #endif //PROJECT_COLOR_H
29
30
31
```

```
1
2 #include "Color.h"
3
4 /*
5  * Constructor
6  */
7 Color::Color(float red, float blue, float green, float clr,uint8_t index){
8     //Arguments: for the corresponding color, output frequencies of the TCS3200 with red-filtered, blue-
9     filtered,
10    //and green-photodiodes divided by the frequency obtained with no filter. Units can be any derived
11    from the Hz,
12    //as long as the 4 arguments have the same unit.
13    //The components of the vector correspond to the output frequencies of the TCS3200 with red-filtered
14    ,
15    // blue-filtered,and green-photodiodes divided by the frequency obtained with no filter.
16    this->redN=red/clr*100;
17    this->blueN=blue/clr*100;
18    this->greenN=green/clr*100;
19    this->index=index;
20
21 }
22
23 /*
24  * distance()
25  *
26  * It allows to compute the Euclidian distance to the vector represented by an instance of Color
27  */
28 double Color::distance(float red, float blue, float green, float clr) {
29
30     red=red/clr*100;
31     blue=blue/clr*100;
32     green=green/clr*100;
33
34     return sqrt(pow(red-this->redN,2)+pow(blue-this->blueN,2)+pow(green-this->greenN,2));
35
36 }
37
38
```

```
1
2 #ifndef PROJECT_TAOS_H
3 #define PROJECT_TAOS_H
4 #include "Arduino.h"
5 #include "Color.h"
6 #include "constantes.h"
7
8 /*
9  * class Taos
10  *
11  * This class represents the light sensor TCS3200. It allows to measure the irradiance
12  * emitted by the observed objects with its different photodiodes, to compare the results
13  * with the colors stored in memory and deduce the color of the observed object
14  */
15 class Taos{
16 public:
17     //ATTRIBUTES
18     //Pins used for the TCS3200
19     int pinS0;
20     int pinS1;
21     int pinS2;
22     int pinS3;
23     int pinOut;
24
25     int N; //Number of colors stored in memory
26     Color *possibleColors; //Matrix containing the N colors stored in memory. Each is represented by an
    object of the
27     //class Color
28
29     //METHODS
30     Taos(uint8_t pinS0, uint8_t pinS1, uint8_t pinS2, uint8_t pinS3, uint8_t pinOut, uint8_t N, Color *
    possibleColors);
31     int readColor();
32     float readFrequency();
33     void init();
34 };
35
36 #endif //PROJECT_TAOS_H
37
38
39
```

```
1
2 #include "Taos.h"
3
4 /*
5  * Constructor
6  */
7 Taos::Taos(uint8_t pinS0, uint8_t pinS1, uint8_t pinS2, uint8_t pinS3, uint8_t pinOut, uint8_t N, Color
  *possibleColors):pinS0(pinS0),pinS1(pinS1),pinS2(pinS2),pinS3(pinS3),
8 pinOut(pinOut),N(N),possibleColors(possibleColors) { }
9
10 /*
11  * init()
12  *
13  * This method allows to setup the pins used to control and read the TCS3200
14  */
15 void Taos::init(){
16
17     pinMode(this->pinS0, OUTPUT);
18     pinMode(this->pinS1, OUTPUT);
19     pinMode(this->pinS2, OUTPUT);
20     pinMode(this->pinS3, OUTPUT);
21     pinMode(this->pinOut, INPUT);
22     digitalWrite(this->pinS0, LOW);
23     digitalWrite(this->pinS1, HIGH);
24
25
26 }
27
28 /*
29  * readFrequency()
30  *
31  * This method returns the frequency of the TCS3200 output signal, in Hz. If the period of the measured
  signal is too
32  * big (greater than TIMEOUT_PULSEIN*2), it returns 0.
33  */
34 float Taos::readFrequency() {
35     unsigned long demiPeriod=pulseIn(this->pinOut, HIGH, TIMEOUT_PULSEIN);
36
37     if(demiPeriod==0){
38         return -1;
39     }
40
41     return (float)500000.0/demiPeriod;
42
43 }
44
45 /*
46  * readColor()
47  *
48  * This method allows to measure a color with the TCS3200. With the results, it computes the Euclidian
  distance of the
49  * measured color to the colors stored in memory (possibleColors) and returns the index of the nearest
  vector.
50  */
51 int Taos::readColor() {
52
53     //Read frequencies with no filter, red filter, blue filter and green filter, respectively.
54     float fClear;
55     float fRed;
56     float fBlue;
57     float fGreen;
58     //We read the output frequencies for the observed object
59     digitalWrite(this->pinS2, HIGH);
60     digitalWrite(this->pinS3, LOW);
61     fClear=readFrequency();
62     if(fClear==-1){
63         return -1;
64     }
65
66     digitalWrite(this->pinS2, LOW);
67     digitalWrite(this->pinS3, LOW);
```

```
68   fRed=readFrequency();
69   if(fRed==-1){
70       return -1;
71   }
72
73   digitalWrite(this->pinS2, LOW);
74   digitalWrite(this->pinS3, HIGH);
75   fBlue=readFrequency();
76   if(fBlue==-1){
77       return -1;
78   }
79
80   digitalWrite(this->pinS2, HIGH);
81   digitalWrite(this->pinS3, HIGH);
82   fGreen=readFrequency();
83   if(fGreen==-1){
84       return -1;
85   }
86
87   //We compute the distance to all possible colors keep the minimal
88   double distance=this->possibleColors[0].distance(fRed, fBlue, fGreen, fClear);
89   int colorIndex=this->possibleColors[0].index;
90   int i;
91
92   for(i=1; i<this->N; i++){
93       if(this->possibleColors[i].distance(fRed, fBlue, fGreen, fClear)<distance){
94           distance=this->possibleColors[i].distance(fRed, fBlue, fGreen, fClear);
95           colorIndex=this->possibleColors[i].index;
96       }
97   }
98
99   //We return the index of the color corresponding to the minimal distance
100   return colorIndex;
101
102
103
104 }
105
106
107
```

```
1 #ifndef SLAVE_PINCE_PINCE_H
2 #define SLAVE_PINCE_PINCE_H
3
4 #include <Arduino.h>
5 #include "Servomotor.h"
6 #include "Potentiometer.h"
7 #include "Taos.h"
8 #include "constantes.h"
9
10 //ENUMS
11
12 //Task the gripper must do (following an order sent by the master)
13 enum Task {
14     NONE,
15     OPEN_COMPLETELY,
16     CLOSE_COMPLETELY,
17     GRAB_SEASHELL,
18     READ_COLOR
19 };
20
21
22 //Action or movement the gripper can perform
23 enum Movement {
24     STOPPED,
25     OPENING,
26     CLOSING,
27     GRABBING_SEASHELL,
28     READING_COLOR,
29     FINISHED_READING_COLOR,
30     ERROR
31 };
32
33 //Possible errors of the system
34 enum Error {
35     NONE_ERR,
36     ERR_TOO_OPEN,
37     ERR_TOO_CLOSE,
38     ERR_TOO_MUCH_INT,
39     ERR_OBSTRUCTION_OPENING,
40     ERR_OBSTRUCTION_CLOSING,
41     ERR_COLOR_SENSOR_NOT_WORKING
42 };
43 };
44 /*
45 * class Gripper
46 *
47 * This class allows to control the whole gripper system.
48 */
49 class Gripper {
50 public:
51
52     //METHODS
53     Gripper(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t pinVout,
54             uint8_t pinPotentiometer,
55             uint8_t pinS0, uint8_t pinS1, uint8_t pinS2, uint8_t pinS3, uint8_t pinOut, uint8_t N, Color
56             *possibleColors);
57     void init();
58     void followOrder(uint8_t order);
59     void updateReal();
60     void openGripper();
61     void closeGripper();
62     void stop();
63     void grabSomething(uint8_t alpha);
64     void readColor();
65
66     //ATTRIBUTES
67     //The gripper includes a servomotor, a potentiometer and a taos (TCS3200 sensor)
68     Servomotor *servomotor;
69     //For our mechanical assembly, if we command sevomotor->writeAlphaCommand(int alphaCommand)
70     // with alphaCommand<0 the griper CLOSES. With alphaCommand>0 it OPENS.
71     Potentiometer *potentiometer;
```

```
70     Taos *taos;
71
72     float intensity; //Intensity flowing through the gripper's servomotor
73     int angle; //Angle of the gripper
74     uint8_t lastReadColor; //Last color detected by the system
75     Movement movement; //Mouvement of the gripper performs at each moment
76     Task task; //Task the gripper follows at each moment
77     bool isGrabbing; //1: the gripper is loaded with an object
78     Error error; //Error at each moment
79 };
80
81 #endif //SLAVE_PINCE_PINCE_H
82
```

```
1 #include "Gripper.h"
2
3
4 /*
5  * Constructor
6  */
7 Gripper::Gripper(uint8_t pinEnable, uint8_t pinInput1, uint8_t pinInput2, uint8_t pinVref, uint8_t
   pinVout, uint8_t pinPotentiometer,
8               uint8_t pinS0, uint8_t pinS1, uint8_t pinS2, uint8_t pinS3, uint8_t pinOut, uint8_t N,
   Color *possibleColors) {
9     potentiometer = new Potentiometer(pinPotentiometer);
10    servomotor = new Servomotor(pinEnable, pinInput1, pinInput2, pinVref, pinVout);
11    taos=new Taos(pinS0,pinS1,pinS2,pinS3,pinOut,N,possibleColors);
12 }
13
14 /*
15  * followOrder()
16  *
17  * This method is called in the loop of the main program,
18  * each time the master sends an order different from 0.
19  * According to the sent variable, the function updates the attributes task and movement.
20  */
21 void Gripper::followOrder(uint8_t order) {
22     switch (order) {
23         case OPEN_COMPLETELY:
24             this->movement = OPENING;
25             this->task = OPEN_COMPLETELY;
26
27             break;
28         case CLOSE_COMPLETELY:
29             this->movement = CLOSING;
30             this->task = CLOSE_COMPLETELY;
31             break;
32         case GRAB_SEASHELL:
33             this->movement = GRABBING_SEASHELL;
34             this->task = GRAB_SEASHELL;
35             break;
36         case READ_COLOR:
37             this->movement = READING_COLOR;
38             this->task = READ_COLOR;
39             break;
40         case NONE:
41             break;
42         default:
43             break;
44     }
45 }
46
47
48 /*
49  * updateReal()
50  *
51  * This method is called at the beginning of each loop of the main program.
52  *
53  * It allows to command different actions (e.g. calling methods as closeGripper()) depending on the
54  * value of movement. It also updates the values of intensity and angle and detects intensity and angle
55  * errors.
56  */
57 void Gripper::updateReal() {
58     //If the movement is other than ERROR, then there's no error in the system
59     if (movement != ERROR)
60         error = NONE_ERR;
61
62     //We update the values of angle and intensity
63     this->angle = potentiometer->getValue();
64     this->intensity = servomotor->readIntensity();
65
66     //We assure these values haven't reached the error limits
67     if (this->angle > ANGLE_ERROR_OPEN && this->movement != CLOSING) {
68         stop();
69         this->error = ERR_TOO_OPEN;
```

```
70     this->movement = ERROR;
71     return;
72 } else if (this->angle < ANGLE_ERROR_CLOSE && this->movement != OPENING) {
73     stop();
74     this->error = ERR_TOO_CLOSE;
75     this->movement = ERROR;
76     return;
77 } else if (this->intensity > INTENSITY_ERROR) {
78     stop();
79     this->error = ERR_TOO_MUCH_INT;
80     this->movement = ERROR;
81     return;
82 }
83
84 //Depending on the value of the attribute movement, an action is commanded
85 switch (movement) {
86     case OPENING:
87         openGripper();
88         break;
89     case CLOSING:
90         closeGripper();
91         break;
92     case STOPPED:
93         stop();
94         break;
95     case GRABBING_SEASHELL:
96         grabSomething(ALPHA_SEASHELL);
97         break;
98     case READING_COLOR:
99         readColor();
100        break;
101    case FINISHED_READING_COLOR:
102        break;
103    case ERROR:
104        this->stop();
105        break;
106 }
107 }
108
109 /*
110  * openGripper()
111  *
112  * This method is called to open the gripper.
113  */
114 void Gripper::openGripper() {
115     //If angle>=ANGLE_LIMIT_OPEN the gripper has finished opening, it stops.
116     if (this->angle >= ANGLE_LIMIT_OPEN) {
117         this->stop();
118         this->movement = STOPPED;
119         return;
120     }
121
122     //If intensity > INTENSITY_THRESHOLD there's an object blocking the opening
123     if (this->intensity > INTENSITY_THRESHOLD) {
124         this->error = ERR_OBSTRUCTION_OPENING;
125         this->movement = ERROR;
126         return;
127     }
128
129     //Else, the gripper continues opening
130     this->servomotor->writeAlphaCommand(ALPHA_OPENING);
131     this->isGrabbing = false; //We can assure the gripper isn't loaded
132 }
133
134 /*
135  * openGripper()
136  *
137  * This method is called to close the gripper, without grabbing an object
138  */
139 void Gripper::closeGripper() {
```

```
141 //If angle<=ANGLE_LIMIT_CLOSE the gripper has finished closing, it stops.
142 if (this->angle <= ANGLE_LIMIT_CLOSE) { //Gripper a fini de se fermer sans rien saisir
143     this->movement = STOPPED;
144     return;
145 }
146
147 //If intensity > INTENSITY_THRESHOLD there's an object blocking the closing
148 if (this->intensity >= INTENSITY_THRESHOLD) {
149     this->error = ERR_OBSTRUCTION_CLOSING;
150     this->movement = ERROR;
151     return;
152 }
153
154 //Else, the gripper continues closing
155 this->servomotor->writeAlphaCommand(ALPHA_CLOSING);
156 this->isGrabbing = false; //We can assure the gripper isn't loaded
157 }
158
159 /*
160 * stop()
161 * This method stops the servomotor. After doing so, attribute task is updated to NONE
162 */
163 void Gripper::stop() {
164     this->servomotor->writeAlphaCommand(0);
165     this->task = NONE;
166 }
167
168 /*
169 * grabSomething(uint8_t alpha)
170 * This method allows to grab an object, specifying with which duty cycle the motor rotation has to be
171 * commanded
172 * When the gripper grabs an object (motor blocked) the duty cycle is related to the torque applied by
173 * the
174 * motor (the greater duty cycle, the greater torque)
175 */
176 void Gripper::grabSomething(uint8_t alpha) { //alpha has to be positif, in this case it's known that the
177     gripper has to
178
179     //close and it's later transformed into negative
180     //If angle<=ANGLE_LIMIT_CLOSE the gripper has finished closing, it stops.
181     if (this->angle <= ANGLE_LIMIT_CLOSE) {
182         this->movement = STOPPED;
183         return;
184     }
185
186     //Else, the gripper "tries" to grab the object
187     this->servomotor->writeAlphaCommand(-alpha);
188     // If intensity > INTENSITY_THRESHOLD an object is gripped
189     this->isGrabbing = this->intensity > INTENSITY_THRESHOLD;
190 }
191
192 /*
193 * readColor()
194 * This method allows to read the color of the object in the gripper and updates the value of
195 * lastReadColor
196 */
197 void Gripper::readColor() {
198     this->lastReadColor=taos->readColor();
199     if(this->lastReadColor==-1){
200         this->error = ERR_COLOR_SENSOR_NOT_WORKING;
201         this->movement = ERROR;
202         return;
203     }
204
205     this->movement=FINISHED_READING_COLOR;
206 }
207 }
```

```
208
209 /*
210  * init()
211  *
212  * This method initializes the attributes of the class, the potentiometer and servomotor objects
213  */
214 void Gripper::init() {
215     this->angle = potentiometer->getValue();
216     this->intensity = servomotor->readIntensity();
217     this->error = NONE_ERR;
218     this->movement = STOPPED;
219     this->task = NONE;
220     this->isGrabbing = false;
221     this->potentiometer->init();
222     this->servomotor->init();
223     this->taos->init();
224 }
225
226
227
228
229
230
231
232
233
234
```


DOCUMENT 2

SUMMARY OF CHARGES

INTRODUCTION

This final document presents the expenses incurred during the development of this project.

The primary goal of the project was for us to improve our knowledge in electronics and other areas. In other words, it had a major pedagogical aim, and it has not been intended to create a future application for the industry. For this reason, the depreciation costs of the equipment and the software used have not been included, these were provided by CentraleSupélec and the LISA. Neither have labor costs been computed, since these working hours are a part of our university education.

It is important to clarify that not all items have been directly purchased by the team (some of them were donated by our sponsors or by the LISA). However, the costs of these components have been included in this summary, in order to have an estimate of the total cost of the project. If the real price of one of these items was not known, the prices proposed by common electronic suppliers in France have been used as a reference.

All prices include taxes.

QUANTITIES

1. ELEVATING SUBSYSTEM

USAGE	ITEM	DESCRIPTION	QUANTITY (units)
ELEVATOR SLAVE CARD	ATmega328p	PDIP	1
	Quartz crystal 16 MHz	Through-hole, 2 pins	1
	Switch (reset)	No-push button, SMD, 4 pins	1
	Connector (ICSP, male)	Male header, 2.54 mm pitch, 2x3	1
	Connector (Power male)	Male terminal block, 5.08 mm pitch, 1x2	1
	Connector (Power female)	Female terminal block, 5.08 mm pitch, 1x3	1
	Connector (Bus male)	Shrouded male header, 2.54 mm pitch, 2x7	1
	Connector (Bus female)	Ribbon crimp connector, 2.54 mm pitch, 2x7	1
	Linear actuator SYS42STH38-1684A Pololu Electronics	28 cm screw	1
	Shield for DRV8825, Pololu Electronics		2
	Cooper clad board	Double sided, 100x160X1.6 mm	1
	Passive components of common value	Resistors, capacitors	6
TESTS	Stepper motor driver DRV8825		2
MECHANICAL STRUCTURE	PVC foam board	500x400x100 mm, white	1
	PLA (3D printer)		30 g
	Smooth rod	Steel, 6 mm diameter, 50 cm length	1
	Threaded rod	Steel, 6 mm diameter, 20 cm length	1
	Tube	Aluminium, 12 mm diameter, 50 cm length	1
	Linear bush	6 mm internal diameter, 12 mm external diameter, 22 mm length	2
	Small mechanical pieces	Screws, nuts, bolts etc.	22

2. GRIPPING SUBSYSTEM

USAGE	ITEM	DESCRIPTION	QUANTITY (units)
SERVO MOTOR CARD AND SPARE SERVO MOTOR CARD	Servo motor 805BB		1
	H-bridge L298N		2
	Schottky diodes BAT60A		8

UNIVERSIDAD PONTIFICIA DE COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECÁNICO – PROYECTO DE FIN DE GRADO

USAGE	ITEM	DESCRIPTION	QUANTITY (units)
	Kelvin resistor	0.01 Ω	2
	Instrumental amplifier MCP6N11		2
	Operational amplifier MCP6021		2
	Decoupling capacitor 1	22 uF, 25V, low ESR	1
	Decoupling capacitor 2	47 uF, 25V, low ESR	2
	Passive components of common value	Resistors and capacitors	24
GRIPPER SLAVE CARD AND SPARE GRIPPER SLAVE CARD	ATmega328p	PDIP	2
	Quartz crystal 16 MHz	Through-hole, 2 pins	2
	Switch (reset)	No-push button, SMD, 4 pins	2
	Connector (ICSP, male)	Male header, 2.54 mm pitch, 2x3	2
	Connector (Bus male)	Shrouded male header, 2.54 mm pitch, 2x7	2
	Connector (Bus female)	Ribbon crimp connector, 2.54 mm pitch, 2x7	2
	Connector (Power male)	Male terminal block, 5.08 mm pitch, 1x2	4
	Connector (Power female)	Female terminal block, 5.08 mm pitch, 1x3	4
	Connector (Ribbon-cable male)	Polarized male connector, 2.54 pitch, 1x8	2
	Connector (Ribbon-cable female)	Polarized female connector, 2.54 pitch, 1x8	2
	Switch voltage regulator LM2679-5.0		2
	Decoupling capacitor	47 uF, 25V	2
	Inductance P0841SNL		2
	Eight-wire ribbon cable		2
	Passive components of common values		32
REDUCED SERVO MOTOR CIRCUIT CARD	Kelvin resistor	0.01 Ω	1
	Instrumental amplifier MCP6N11		1
	Operational amplifier MCP6021		1
	Passive components of common value	Resistors and capacitors	2
ALL CARDS	Cooper clad board	Double sided, 100x160X1.6 mm	2
TESTS	H-bridge L298N		1
	Double schottky diodes MBR3050CT	MBR3050CT, DOUBLE, CATHODE COMMUNE,	2
	Plywood board	600x600x18 mm	1

USAGE	ITEM	DESCRIPTION	QUANTITY (units)
MECHANICAL STRUCTURE	Small mechanical pieces	Screws, nuts, bolts etc.	24

3. COLOR-SENSING SUBSYSTEM

USAGE	ITEM	DESCRIPTION	QUANTITY (units)
PROTOTYPE CARD	Color sensor TCS3200		1
	White LED multicom 703-1026		1
	Passive components of common values		2
	Cooper clad board	Double sided, 100x160X1.6 mm	1
PROTOTYPE CASE	PLA (3D printer)		15 g
TESTS	Color sensor TCS3200		1

4. OTHER MATERIALS

ITEM	DESCRIPTION	QUANTITY (units)
Arduino Uno Board		1
Breadboard		1
Welding wire	250 g, 80 to 100% tin	1
Coil of wire for breadboards	50m	1

UNIT PRICES

1. ELEVATING SUBSYSTEM

USAGE	ITEM	PRICE PER UNIT (€)
ELEVATOR SLAVE CARD	ATmega328p	3,040
	Quartz crystal 16 MHz	0,407
	Switch (reset)	0,091
	Connector (ICSP, male)	0,774
	Connector (Power male)	0,372
	Connector (Power female)	1,148
	Connector (Bus male)	0,598
	Connector (Bus female)	0,636
	Linear actuator SYS42STH38-1684A Pololu Electronics	57,960
	Shield for DRV8825, Pololu Electronics	9,900
	Cooper clad board	2,000
	Passive components of common value	0,200
TESTS	Stepper motor driver DRV8825	5,320
MECHANICAL STRUCTURE	PVC foam board	6,800
	PLA (3D printer)	14 €/300 g
	Smooth rod	1,600
	Threaded rod	2,350
	Tube	2,300
	Linear bush	14,830
	Small mechanical pieces	0,200

2. GRIPPING SUBSYSTEM

USAGE	ITEM	PRICE PER UNIT (€)
SERVO MOTOR CARD AND SPARE SERVO MOTOR CARD	Servo motor 805BB	44,900
	H-bridge L298N	5,050
	Schottky diodes BAT60A	0,121
	Kelvin resistor	1,860
	Instrumental amplifier MCP6N11	1,253
	Operational amplifier MCP6021	0,942
	Decoupling capacitor 1	0,316
	Decoupling capacitor 2	0,642
	Passive components of common value	0,200
GRIPPER SLAVE CARD AND SPARE GRIPPER SLAVE CARD	ATmega328p	3,040
	Quartz crystal 16 MHz	0,407
	Switch (reset)	0,091
	Connector (ICSP, male)	0,774
	Connector (Bus male)	0,636
	Connector (Bus female)	0,598

USAGE	ITEM	PRICE PER UNIT (€)
	Connector (Power male)	1,148
	Connector (Power female)	0,372
	Connector (Ribbon-cable male)	0,301
	Connector (Ribbon-cable female)	0,508
	Switch voltage regulator LM2679-5.0	5,282
	Decoupling capacitor	0,642
	Inductance P0841SNL	2,740
	Eight-wire ribbon cable	1,200
	Passive components of common values	0,200
REDUCED SERVO MOTOR CIRCUIT CARD	Kelvin resistor	1,860
	Instrumental amplifier MCP6N11	1,235
	Operational amplifier MCP6021	0,942
	Passive components of common value	0,200
ALL CARDS	Cooper clad board	2,000
TESTS	H-bridge L298N	5,050
	Double schottky diodes MBR3050CT	0,732
MECHANICAL STRUCTURE	Plywood board	9,650
	Small mechanical pieces	0,200

3. COLOR-SENSING SUBSYSTEM

USAGE	ITEM	PRICE PER UNIT (€)
PROTOTYPE CARD	Color sensor TCS3200	4,240
	White LED multicom 703-1026	0,236
	Passive components of common values	0,200
	Cooper clad board	2,000
PROTOTYPE CASE	PLA (3D printer)	14 €/300 g
TESTS	Color sensor TCS3200	4,240

4. OTHER MATERIALS

ITEM	PRICE PER UNIT (€)
Arduino Uno Board	22,450
Breadboard	10,000
Welding wire	30,000
Coil of wire for breadboards	10,000

PARTIAL AMOUNTS

1. ELEVATING SUBSYSTEM

USAGE	ITEM	QUANTITY (units)	PRICE PER UNIT	COST (€)
ELEVATOR SLAVE CARD	ATmega328p	1	3,040	3,040
	Quartz crystal 16 MHz	1	0,407	0,407
	Switch (reset)	1	0,091	0,091
	Connector (ICSP, male)	1	0,774	0,774
	Connector (Power male)	1	0,372	0,372
	Connector (Power female)	1	1,148	1,148
	Connector (Bus male)	1	0,598	0,598
	Connector (Bus female)	1	0,636	0,636
	Linear actuator SYS42STH38-1684A Pololu Electronics	1	57,960	57,960
	Shield for DRV8825, Pololu Electronics	2	9,900	19,800
	Cooper clad board	1	2,000	2,000
	Passive components of common value	6	0,200	1,200
TESTS	Stepper motor driver DRV8825	2	5,320	10,640
MECHANICAL STRUCTURE	PVC foam board	1	6,800	6,800
	PLA (3D printer)	30 g	14 €/300 g	1,400
	Smooth rod	1	1,600	1,600
	Threaded rod	1	2,350	2,350
	Tube	1	2,300	2,300
	Linear bush	2	14,830	29,660
	Small mechanical pieces	22	0,200	4,400

2. GRIPPING SUBSYSTEM

USAGE	ITEM	QUANTITY (units)	PRICE PER UNIT	COST (€)
SERVO MOTOR CARD AND SPARE SERVO MOTOR CARD	Servo motor 805BB	1	44,900	44,900
	H-bridge L298N	2	5,050	10,100
	Schottky diodes BAT60A	8	0,121	0,968
	Kelvin resistor	2	1,860	3,720
	Instrumental amplifier MCP6N11	2	1,253	2,506
	Operational amplifier MCP6021	2	0,942	1,884
	Decoupling capacitor 1	1	0,316	0,316
	Decoupling capacitor 2	2	0,642	1,284
	Passive components of common value	24	0,200	4,800
GRIPPER SLAVE CARD AND SPARE GRIPPER SLAVE CARD	ATmega328p	2	3,040	6,080
	Quartz crystal 16 MHz	2	0,407	0,814
	Switch (reset)	2	0,091	0,182
	Connector (ICSP, male)	2	0,774	1,548
	Connector (Bus male)	2	0,636	1,272

UNIVERSIDAD PONTIFICIA DE COMILLAS
ESCUELA TÉCNICA SUPERIOR DE INGENIERÍA (ICAI)
INGENIERO ELECTROMECÁNICO – PROYECTO DE FIN DE GRADO

USAGE	ITEM	QUANTITY (units)	PRICE PER UNIT	COST (€)
	Connector (Bus female)	2	0,598	1,196
	Connector (Power male)	4	1,148	4,592
	Connector (Power female)	4	0,372	1,488
	Connector (Ribbon-cable male)	2	0,301	0,602
	Connector (Ribbon-cable female)	2	0,508	1,016
	Switch voltage regulator LM2679-5.0	2	5,282	10,564
	Decoupling capacitor	2	0,642	1,284
	Inductance P0841SNL	2	2,740	5,480
	Eight-wire ribbon cable	2	1,200	2,400
	Passive components of common values	32	0,200	6,400
REDUCED SERVO MOTOR CIRCUIT CARD	Kelvin resistor	1	1,860	1,860
	Instrumental amplifier MCP6N11	1	1,235	1,235
	Operational amplifier MCP6021	1	0,942	0,942
	Passive components of common value	2	0,200	0,400
ALL CARDS	Cooper clad board	2	2,000	4,000
TESTS	H-bridge L298N	1	5,050	5,050
	Double schottky diodes MBR3050CT	2	0,732	1,464
MECHANICAL STRUCTURE	Plywood board	1	9,650	9,650
	Small mechanical pieces	24	0,200	4,800

3. COLOR-SENSING SUBSYSTEM

USAGE	ITEM	QUANTITY (units)	PRICE PER UNIT	COST (€)
PROTOTYPE CARD	Color sensor TCS3200	1	4,240	4,240
	White LED multicom 703-1026	1	0,236	0,236
	Passive components of common values	2	0,200	0,400
	Cooper clad board	1	2,000	2,000
PROTOTYPE CASE	PLA (3D printer)	15 g	14 €/300 g	0,700
TESTS	Color sensor TCS3200	1	4,240	4,240

4. OTHER MATERIALS

ITEM	QUANTITY (units)	PRICE PER UNIT	COST (€)
Arduino Uno Board	1	22,450	22,450
Breadboard	1	10,000	10,000
Welding wire	1	30,000	30,000
Coil of wire for breadboards	1	10,000	10,000

TOTAL COSTS

	COST (€)
ELEVATING SUBSYSTEM	147,176
GRIPPING SUBSYSTEM	144,797
COLOR-SENSING SUBSYSTEM	11,816
OTHER MATERIALS	72,450
TOTAL PROJECT COST	376,239