Trabajo Fin de Máster
Máster en Ingeniería Aeronáutica

Diseño y puesta en marcha de una práctica experimental sobre apuntamiento de una cámara

Autor: Daniel Enrique Martínez Coza
Tutor: Rafael Vázquez Valenzuela

Dep. Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla

Sevilla, 2018

Trabajo Fin de Máster
Máster en Ingeniería Aeronáutica

Diseño y puesta en marcha de una práctica experimental sobre apuntamiento de una cámara

Autor:
Daniel Enrique Martínez Coza

Tutor:
Rafael Vázquez Valenzuela
Profesor titular

Dep. de Ingeniería Aeroespacial y Mecánica de Fluidos
Escuela Técnica Superior de Ingeniería
Universidad de Sevilla
Sevilla, 2018

Trabajo Fin de Máster: Diseño y puesta en marcha de una práctica experimental sobre apuntamiento de una cámara

Autor: Daniel Enrique Martínez Coza

Tutor: Rafael Vázquez Valenzuela

El tribunal nombrado para juzgar el Proyecto arriba indicado, compuesto por los siguientes miembros:

Presidente:

Vocales:

Secretario:

Acuerdan otorgarle la calificación de:

Sevilla, 2018

El Secretario del Tribunal

A todos los que alguna vez me han ayudado y orientado.

Agradecimientos

En primer lugar he de agradecer a mi tutor Rafael Vázquez Valenzuela por ofrecerme la oportunidad de realizar este proyecto y por depositar en mí su confianza.

Por otro lado agradecer a mi familia, que nunca me ha fallado y ha hecho posible que realice este proyecto al brindarme la oportunidad de estudiar y darme su incondicional apoyo.

Finalmente, dar gracias a todos los que me han ayudado a crecer y madurar en mi camino.

Daniel Enrique Martínez Coza

Sevilla, 2018

Resumen

El objetivo de este proyecto es el desarrollo de una práctica experimental sobre apuntamiento de una cámara que permita ilustrar la cinemática de la actitud y las maniobras que se realizan para modificarla. Para ello, se partirá inicialmente de los conceptos básicos de la cinemática de la actitud. Haciendo uso de un controlador PID y de un microcontrolador Arduino UNO se desarrollarán algoritmos de detección de elementos y de trayectorias que actúan modificando las posiciones de los servomotores de la cámara. La práctica se desarrollará con estos algoritmos y con interfaces de usuario creadas en MATLAB. Finalmente, se propondrán mejoras y líneas de trabajo futuras.

Abstract

The main purpose of this Master's Thesis is to develop an experimental practice about a tracking camera device that allows to illustrate the attitude kinematics and manoeuvres. To do so, the thesis starts with basic concepts of attitude kinematics. Then, using a PID controller and an Arduino UNO board, algorithms are developed to detect elements and to create trajectories. These algorithms work by changing the servomotors' positions. A practical lab for student is developed, which makes use of these algorithms and graphical interfaces created in MATLAB. To conclude the work, some possible improvements and extensions to the thesis are proposed.

Índice

ÍNDICE DE TABLAS

ÍNDICE DE FIGURAS

Notación

$\vec{\omega}$	Vector velocidad angular
$\vec{e}^{S}_{S/S'}, \vec{e}_{S/S'}, \vec{e}$	Eje de Euler
θ	Ángulo de Euler
e_i	Componentes del eje de Euler
$\vec{\theta}$	Vector de rotación
z	Número complejo
q	Cuaternión
q_i	Componentes del cuaternión
M	Par aplicado
I	Inercia alrededor del eje de giro
T	Duración de la maniobra
α	Fracción de la duración de la maniobra de aceleración
$u(t)$	Salida del controlador PID
$e(t)$	Error del controlador PID
K_p	Constante proporcional del PID
K_i	Constante integral del PID
K_d	Constante derivativa del PID

1. INTRODUCCIÓN

La mayor parte de los vehículos espaciales poseen elementos orientables con fines diversos. Es por ello que es de vital importancia conocer la actitud de estos vehículos y poder modificarla para alcanzar la actitud deseada. El conocimiento de la actitud de un vehículo puede ser conocida haciendo uso de múltiples sensores: sensores inerciales, sensores magnéticos y sensores ópticos. El uso de sensores ópticos está muy extendido, siendo el sensor más preciso uno de ellos: el sensor de estrellas.

El tiempo o el consumo energético pueden ser unas variables de vital importancia en el desempeño de las operaciones de un vehículo espacial y es por ello que las trayectorias seguidas para modificar la actitud toman un papel de gran importancia. Existen trayectorias que optimizan estos requisitos, pero para ello dependen del conocimiento de su actitud inicial. Es la conjunción de la teoría de la cinemática de la actitud, los requisitos operacionales y los sensores lo que permite la existencia de las operaciones de vehículos en el espacio.

1.1 Objetivos y alcance del proyecto

El objetivo principal de este proyecto es evaluar y desarrollar una posible práctica que ejemplifique de forma aplicada giros y capturas de blanco, para la asignatura *Dinámica de Vehículos Espaciales* del Máster en Ingeniería Aeronáutica impartido en la Escuela Técnica Superior de Ingeniería de la Universidad de Sevilla. Para ello se hará uso de una cámara Pixy de bajo coste con capacidad de detección de objetivos en base al color y que cuenta con un mecanismo pan/tilt que permite su rotación con 2 grados de libertad.

Como objetivo último se diseñará una práctica en la que se empleará la cámara Pixy para mostrar de forma didáctica distintos tipos de giros y un procesado simple de imágenes que ejemplifique el tratamiento de imagen empleado por satélites.

Para ello, será necesario el desarrollo de un algoritmo de rastreo que sea capaz de barrer un área determinada y detectar los distintos elementos que allí se encuentren para determinar sus posiciones. Además, será necesario hacer uso de un microcontrolador que permita enviar órdenes al sensor Pixy y recibir información para su procesamiento.

Para el desarrollo del algoritmo y el control se ha empleado la plataforma de hardware libre Arduino, ya que se trata de un microcontrolador robusto y posee un entorno de desarrollo software muy sencillo y de fácil uso para la práctica.

Finalmente, es posible englobar los objetivos del proyecto en los siguientes:

- Modelado matemático de la posición de la cámara en función de los grados de libertad de los servomotores.
- Implementación de un algoritmo de rastreo en Arduino que permita la detección y el posicionado de elementos.
- Establecimiento de un método de envío de la información obtenida del rastreo a un ordenador para posterior procesado.
- Procesado de la información obtenida del rastreo empleando la herramienta *MATLAB*.
- Creación de interfaces de usuario para el tratado de la información durante la práctica.
- Generación de archivos de órdenes con las diversas trayectorias para ser ejecutado por el sensor a través de Arduino.

1.2 Contenido y estructura del documento

Una vez establecidos los objetivos y el alcance del proyecto, restaría únicamente establecer el contenido y la estructura del documento. Una vez finalizado el capítulo introductorio, el contenido de los capítulos siguientes sería el que se presenta a continuación:

- ***2. Cinemática de la actitud***: En este capítulo se presenta la base teórica de la cinemática de la actitud. Se exponen métodos de representación de la actitud y maniobras óptimas de actitud básicas en torno a un eje de simetría.
- ***3. Sensor PIXY***: En este capítulo se presentan las características del sensor Pixy, una guía de montaje del conjunto y se exponen las características y funcionalidades de la aplicación para ordenador PixyMon.
- ***4. Arduino***: En este capítulo se presenta una descripción del hardware y del software de Ardunio y se expone la compatibilidad entre el sensor Pixy y Arduino.
- ***5. Controlador PID***: En este capítulo se explica el funcionamiento del controlador PID, se particulariza para este proyecto y se adapta para Arduino IDE.
- ***6. Detección de elementos***: En este capítulo se desarrolla un modelo matemático del movimiento del sensor Pixy y del campo visual del sensor. Además se presenta el algoritmo de detección de elementos empleado.
- ***7. Implementación de trayectorias***: En este capítulo se adaptan las ecuaciones de giros del capítulo 2 a los parámetros de control que se emplearán y se presenta el algoritmo de trayectorias empleado.
- ***8. Procesamiento de datos e Interfaces***: En este capítulo se presenta la aplicación que permite obtener información del sensor Pixy a través del puerto serial y se muestran las interfaces de usuario desarrolladas.
- ***9. Práctica***: En este capítulo se presenta el montaje de la práctica y una guía esquemática de la misma.
- ***10. Conclusiones***: En este capítulo se presentan las conclusiones tras el desarrollo del proyecto y posibles líneas de trabajo futuras junto con mejoras.
- ***Apéndice A***: En este apéndice se encuentra la memoria de la práctica para la asignatura *Dinámica de Vehículos Espaciales.*
- ***Apéndice B***: En este apéndice se encuentran los códigos desarrollados en Arduino IDE para la práctica.
- ***Apéndice C***: En este apéndice se encuentran los códigos de MATLAB desarrollados para la práctica.

2. CINEMÁTICA DE LA ACTITUD

La actitud de un cuerpo puede definirse como su orientación relativa a un cierto sistema de referencia, por lo tanto, si se supone un cuerpo como sólido rígido es posible expresar esta actitud como la orientación de los ejes cuerpo (sistema de referencia solidario al cuerpo) respecto a otro sistema de referencia de interés. Para el caso de cuerpos en órbita, los ejes órbita son un buen ejemplo de sistema de referencia de interés.

La cinemática de la actitud puede describirse como el conjunto de relaciones (en forma de ecuaciones diferenciales) entre la velocidad angular del cuerpo $\vec{\omega}$ y su actitud (que puede ser representada con distintos métodos).

2.1 Métodos de representación de la actitud

Tal y como se ha mencionado anteriormente, si se considera un sólido rígido, únicamente es necesario determinar la orientación de los ejes cuerpo respecto a otro sistema de referencia. Por lo tanto, bajo la hipótesis de sólido rígido, es posible relacionar ambos sistemas de referencia haciendo uso de una rotación y una traslación. Dado el alcance y objetivos del proyecto, la traslación no será de interés y por lo tanto no se incluirá en las relaciones que seguirán.

2.1.1 Ángulo y eje de Euler

El Teorema de Euler establece que: "el movimiento más general posible de un sólido con un punto fijo es una rotación alrededor de un único eje".

Por lo tanto, es posible representar la actitud de un sistema de referencia S respecto a otro sistema de referencia S' haciendo uso únicamente de un eje y un ángulo. Este eje y este ángulo serían:

- Eje de Euler: $\vec{e}_{S/S'}$
- Ángulo de Euler: θ

El eje de Euler se trata de un vector unitario, por lo que se puede escribir: $\vec{e}^{\,S}_{S/S} = [e_x \quad e_y \quad e_z]^T$, donde debe cumplirse: $e_x^2 + e_y^2 + e_z^2 = 1$.

Esto se traduce en que el eje de Euler es la dirección entorno a la que se debe girar un cierto ángulo para hacer coincidir el sistema de referencia S con el sistema de referencia S'. De esta forma, la actitud queda representada con los parámetros $(\vec{e}_{S/S'}, \theta)$.

Esta forma de representar la actitud se caracteriza por cuatro parámetros: las tres componentes del eje de Euler y el ángulo de giro. De modo que se podría construir un vector de rotación tal que: $\vec{\theta} = \theta\vec{e}_{S/S'}$. Esta representación permite una interpretación física sencilla, ya que establecería la velocidad angular que habría de mantenerse durante un segundo para pasar de la actitud de referencia a la actitud actual.

Sin embargo, a pesar de su sencilla interpretación física, esta representación presenta ciertas ambigüedades e indefiniciones. Para una cierta actitud dada por $(\vec{e}_{S/S'}, \theta)$, la actitud $(-\vec{e}_{S/S'}, 2\pi - \theta)$ es exactamente idéntica, por lo que sería necesario restringir θ al intervalo $[0, \pi)$. Por otro lado, un giro de 0 y de 2π radianes serían físicamente iguales, pero para el primero resultaría $\vec{\theta} = \vec{0}$ y el para el segundo no se encontraría definido.

2.1.2 Cuaterniones

Los cuaterniones son una creación de Hamilton (siglo XIX) y se trata de una extensión de los números complejos a 4 dimensiones.

Un número complejo se trata de una extensión del conjunto de los números reales por la adición de la unidad imaginaria i tal que $i^2 = -1$, de forma que el número complejo z puede escribirse como:

$$z = x + iy$$

Donde x es la componente real e y la componente imaginaria.

Los números complejos con módulo unidad ($\|z\| = 1$) pueden interpretarse como una rotación plana y pueden escribirse como:

$$z = e^{i\theta} = \cos\theta + i\sin\theta$$

Es posible escribir un cuaternión genérico como:

$$q = q_0 + iq_1 + jq_2 + kq_3$$

Donde q_0 se denomina "parte escalar" y $\vec{q} = [q_1 \quad q_2 \quad q_3]^T$ se denomina "parte vectorial". Pudiendo expresarse el cuaternión como:

$$q = \begin{bmatrix} q_0 \\ \vec{q} \end{bmatrix}$$

A continuación se enumeran las propiedades del álgebra de los cuaterniones:

- **Suma**: la suma es componente a componente. Dado los dos cuaterniones siguientes $q = q_0 + iq_1 + jq_2 + kq_3$ y $q' = q'_0 + iq'_1 + jq'_2 + kq'_3$, se tendría que:

$$q'' = q + q' = (q_0 + q'_0) + i(q_1 + q'_1) + j(q_2 + q'_2) + k(q_3 + q'_3)$$

- **Producto**: el producto se realiza componente a componente, conociendo las siguientes reglas de multiplicación:

$$i \star i = j \star j = k \star k = -1$$

$$i \star j = k, \qquad j \star i = -k$$

$$i \star k = -j, \qquad k \star i = j$$

$$j \star k = i, \qquad k \star j = -i$$

Es posible escribir el producto $q'' = q' \star q$ en forma matricial:

$$\begin{bmatrix} q''_0 \\ q''_1 \\ q''_2 \\ q''_3 \end{bmatrix} = \begin{bmatrix} q'_0 & -q'_1 & -q'_2 & -q'_3 \\ q'_1 & q'_0 & -q'_3 & q'_2 \\ q'_2 & q'_3 & q'_0 & -q'_1 \\ q'_3 & -q'_2 & q'_1 & q'_0 \end{bmatrix} \begin{bmatrix} q_0 \\ q_1 \\ q_2 \\ q_3 \end{bmatrix}$$

- **Conjugado**: se construye de igual forma que con los números complejos. Dado el cuaternión q= $q_0 + iq_1 + jq_2 + kq_3$, se define su conjugado como:

$$q^* = q_0 - iq_1 - jq_2 - kq_3$$

- **Módulo**: se define el módulo de $q = q_0 + iq_1 + jq_2 + kq_3$ como:

$$|q|^2 = q \star q^* = q_0^2 + q_1^2 + q_2^2 + q_3^2$$

- **División**: se define la división usando el conjugado:

$$q' * \frac{1}{q} = \frac{q'}{q} \star \frac{q^*}{q^*} = \frac{(q' \star q^*)}{|q|^2}$$

Dada la actitud de un sistema de referencia a través del eje de Euler y del ángulo de Euler $(\vec{e}, \theta)$, es posible codificar dicha actitud en forma de cuaterniones mediante las relaciones siguientes:

$$q = \begin{bmatrix} \cos(\theta/2) \\ \vec{e}\,\sin(\theta/2) \end{bmatrix}, \qquad |q| = 1$$

Esta representación de la actitud es la más eficiente computacionalmente. Sin embargo, los cuaterniones presentan la misma ambigüedad que el ángulo y eje de Euler, q y $-q$ representan la misma actitud. Además, es una representación matemática sin sentido físico.

2.1.3 Cuaternión de interpolación

Dados dos cuaterniones q_0 y q_1 podría plantearse la cuestión siguiente: ¿es posible construir un cuaternión de interpolación continuo $q(s)$ de forma que $q(0) = q_0$ y $q(1) = q_1$? Es decir, es necesario determinar la distribución de cuaterniones intermedios que representan las actitudes intermedias entre q_0 y q_1.

Para ello, es necesario determinar el cuaternión q_2 que representaría la actitud entre q_0 y q_1. Por lo tanto, el cuaternión de actitud buscado sería aquél que cumple la relación siguiente:

$$q_1 = q_0 \star q_2$$

Aplicando el álgebra de cuaterniones es posible obtener el cuaternión de actitud:

$$q_2 = \frac{1}{q_0} \star q_1 = q_0^* \star q_1 = \begin{bmatrix} \cos(\theta/2) \\ \vec{e}\,\sin(\theta/2) \end{bmatrix}$$

Como se vio anteriormente, un cuaternión codifica en sus componentes un ángulo y un eje de Euler, por lo que, el cuaternión de actitud q_2 representaría un eje y una cierta cantidad de ángulo girado. Por lo tanto, si se varía este ángulo entre $0\ y\ \theta$ podría obtenerse cada una de las actitudes intermedias.

Finalmente, la distribución de actitudes podría expresarse como:

$$q(s) = q_0 \star q_i = q_0 \star \begin{bmatrix} \cos(s\theta/2) \\ \vec{e}\,\sin(s\theta/2) \end{bmatrix}, \qquad donde\ s\epsilon[0,1]$$

En la figura siguiente quedaría ilustrado de forma esquemática y simplificada el proceso anterior:

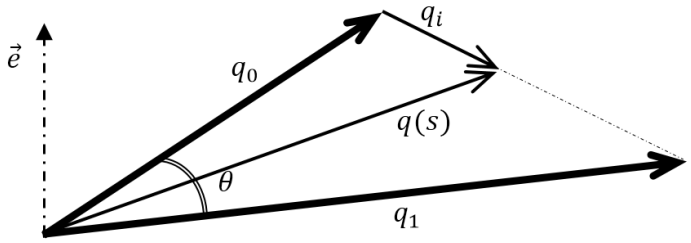


Figura 2-1. Esquema del cuaternión de interpolación

2.2 Maniobras óptimas de actitud básicas en torno a un eje de simetría

En este apartado se mostrarán las trayectorias de tiempo mínimo y de mínima energía obtenidas de la referencia [1]. En esta sección se responde al problema siguiente: dada una actitud inicial A (dada por el cuaternión q_0) y una actitud final B (dada por el cuaternión q_1), se pretende rotar un cuerpo desde la actitud A hasta la B. Esta rotación pretende que tanto al inicio como al final de la maniobra, el cuerpo se encuentre en estado de reposo.

Haciendo uso del concepto de cuaternión de interpolación mencionado en la sección anterior, podemos obtener el camino más corto entre A y B. Simplemente habría que determinar el cuaternión de interpolación, y de dicho cuaternión sería posible obtener el eje entorno al cual sería necesario realizar la rotación y el ángulo que sería necesario girar.

$$q_R = q_0^* \star q_1 = \begin{bmatrix} \cos(\theta/2) \\ \vec{e}\,\sin(\theta/2) \end{bmatrix}$$

Una vez determinada la dirección de giro, sería posible establecer la velocidad angular en cada instante de tiempo:

$$\vec{\omega} = \omega(t)\vec{e}$$

Por lo tanto, dadas una actitud inicial y una final, el problema se vería reducido a determinar la variación de la velocidad angular de forma que se satisfagan las condiciones impuestas. Para ello, se recurrirá a una dinámica simplificada bajo la hipótesis de que el eje de giro es uno de simetría de masa. Por lo tanto:

$$\dot{\theta} = \omega$$

$$I\dot{\omega} = M$$

Bajo las condiciones de contorno siguientes:

$$\theta(0) = 0, \qquad \theta(T) = \theta_F$$

$$\omega(0) = 0, \qquad \omega(T) = 0$$

Donde M es el par aplicado en cada instante, I es la inercia entorno al eje de giro y T es el tiempo de duración de la maniobra.

2.2.1 Maniobra de tiempo mínimo

Para obtener una maniobra de tiempo mínimo es necesario aplicar el máximo par disponible en dos etapas: una etapa de aceleración y otra de desaceleración. En un principio, se supondrá que el par aplicado durante la aceleración (M_{acel}) no será necesariamente el mismo que durante la deceleración (M_{decel}). Es necesario mencionar que el par de aceleración tendrá valor positivo y que el par de deceleración tendrá valor negativo.

Debido a esta ausencia de simetría, la duración de la aplicación de cada par será distinta. Por ello es necesario introducir el parámetro α que indica la cantidad de T que corresponde a la aplicación de (M_{acel}), siendo esta duración αT.

Por lo tanto, el valor del par es:

$$M(t) = \begin{cases} M_{acel}, & 0 \le t \le \alpha T \\ M_{decel}, & \alpha T \le t \le T \end{cases}$$

Integrando la función anterior se obtendría la velocidad angular siguiente:

$$\omega(t) = \begin{cases} \frac{M_{acel}}{I} t, & 0 \le t \le \alpha T \\ \frac{M_{acel}}{I} \alpha T + \frac{M_{decel}}{I} (t - \alpha T), & \alpha T \le t \le T \end{cases}$$

Integrando la velocidad angular se obtendría la evolución del ángulo:

$$\theta(t) = \begin{cases} \frac{M_{acel}}{I} \frac{t^2}{2} , & 0 \le t \le \alpha T \\ \frac{M_{acel}}{I} \frac{(\alpha T)^2}{2} + \frac{M_{acel}}{I} \alpha T (t - \alpha T) + \frac{M_{decel}}{I} \frac{(t - \alpha T)^2}{2}, & \alpha T \le t \le T \end{cases}$$

Si se particulariza la velocidad angular en el instante T es posible establecer el valor del parámetro α como función de los pares aplicados.

$$\omega(T) = \frac{M_{acel}}{I} \alpha T + \frac{M_{decel}}{I} (1 - \alpha) T = \frac{T}{I} (\alpha M_{acel} + (1 - \alpha) M_{decel}) = 0$$

Obteniéndose:

$$\alpha = \frac{-M_{decel}}{M_{acel} - M_{decel}}$$

Si se particulariza el ángulo en el instante T se obtendría la expresión siguiente:

$$\theta_F = \frac{T^2}{2I} \frac{-M_{acel} M_{decel}}{(M_{acel} - M_{decel})}$$

Donde podemos determinar la duración de la trayectoria en función de la inercia y los pares aplicados:

$$T = \sqrt{\frac{2I(M_{acel} - M_{decel})\theta_F}{-M_{acel} M_{decel}}}$$

Si se sustituyen los pares de aceleración y de deceleración por sus valores extremos, se obtendría el tiempo mínimo de maniobra:

$$T_{min} = \sqrt{\frac{2I(M_{max} - M_{min})\theta_F}{-M_{max} M_{min}}}$$

2.2.2 Maniobra de mínima energía

Para obtener una maniobra de mínima energía es necesario minimizar $\int_0^T M^2(\tau) d\tau$ para una duración de la maniobra T dada. Si no existen restricciones, es posible deducir a través del cálculo de variaciones que el par que minimiza la energía total consumida es lineal, por lo que se obtendría una velocidad angular cuadrática:

$$\omega(t) = A + Bt + Ct^2$$

Sujeto a las restricciones siguientes:

$$\omega(0) = \omega(T) = 0$$

De las que se puede obtener:

$$A = 0, \qquad B = -CT$$

Por lo tanto, la velocidad angular quedaría en función de una única incógnita: C.

$$\omega(t) = Ct(t - T)$$

Si se integra la función anterior se obtendría la evolución del ángulo siguiente:

$$\theta_F = \int_0^T \omega(\tau)d\tau = C\int_0^T \tau(\tau - T)d\tau = -C\frac{T^3}{6}$$

Por lo que sería posible determinar la última incógnita como función de la posición final conocida:

$$C = -\frac{6\theta_F}{T^3}$$

Finalmente, las evoluciones temporales del par, la velocidad angular y la posición resultarían:

$$M(t) = \frac{6I\theta_F}{T^3}(T - 2t)$$

$$\omega(t) = \frac{6\theta_F}{T^3}t(T - t)$$

$$\theta(t) = \frac{\theta_F}{T^3}t^2(3T - 2t)$$

3. SENSOR PIXY

Pixy es un sensor de visión rápida ideado para aplicaciones de robótica. Este sensor es capaz de almacenar información de hasta 7 colores diferentes y de detectarlos dentro de su campo de visión. Es capaz de detectar y seguir múltiples blancos de forma simultánea y de proporcionar información de la posición de cada objeto en tiempo real.

Pixy además cuenta con un sistema pan/tilt que le permite orientarse usando dos servomotores que le provee de dos grados de libertad y un amplio campo visual. Pixy tiene la capacidad de enviar órdenes directas a los servomotores para modificar su posición de forma rápida y eficaz. Además, es capaz de cargar programas desde microcontroladores para realizar actividades de seguimiento y posicionado de objetivos.

Es posible enumerar una serie de ventajas del sensor Pixy [2]:

- Sensor de pequeño tamaño.
- Sensor de bajo coste.
- Detecta objetos de un tamaño mínimo de 4x1 pixeles en la imagen.
- Envía información de los blancos detectados 50 veces por segundo.
- Conectividad con Arduino y otros controladores de software libre.
- Librerías para controladores y software en código abierto.
- Aplicación para configurar Pixy gratuita para Windows, MacOS y Linux.

En la figura siguiente es posible observar el sensor Pixy instalado con los dos servomotores:

Figura 3-1. Sensor Pixy instalado con los servomotores y los apoyos

3.1 Características del sensor Pixy y de los servomotores

3.1.1 Características del sensor Pixy

El sensor Pixy cuenta con las siguientes características técnicas [3]:

- **Procesador**: NXP LPC4330, 204 MHz, dual core.
- **Sensor de imagen**: Omnivision OV9715, 1/4“, 1280×800.
- **Campo visual de la lente**: 75 grados en horizontal y 47 grados en vertical.
- **Tipo de lente**: estándar M12.
- **Consumo**: típicamente 140 mA.
- **Alimentación**: USB (5V) o entrada no regulada (6V a 10V).
- **Memoria**: RAM (264K bytes) y Flash (1M bytes).
- **Salidas de datos disponibles**: serial UART, SPI, 12C, USB, digital y analógica
- **Dimensiones**: 2.1 x 2.0 x 1.4 pulgadas.
- **Peso**: 27 gramos.

En la figura siguiente se puede observar la parte posterior del sensor Pixy:

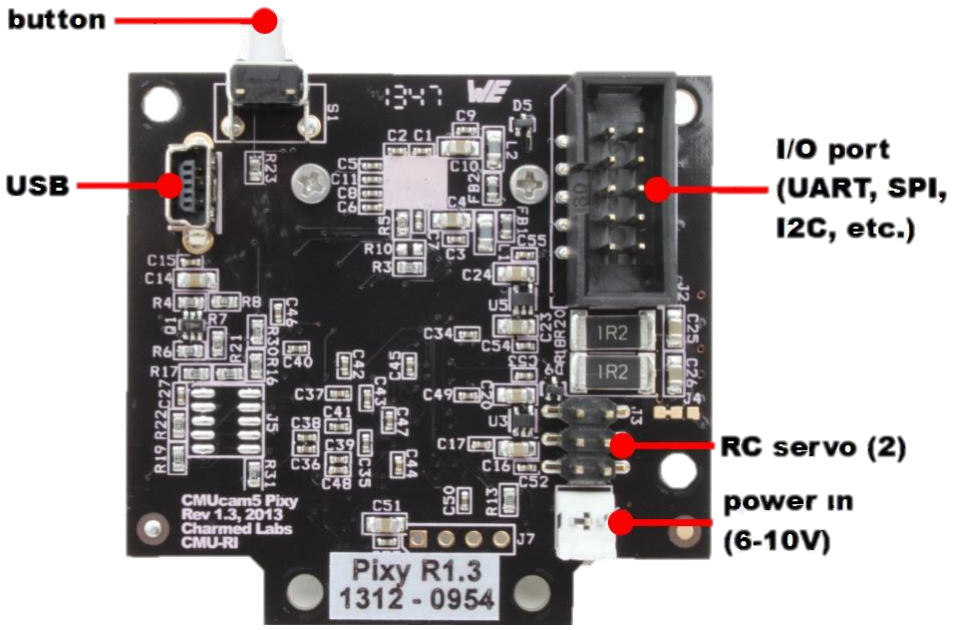


Figura 3-2. Vista posterior del sensor Pixy

Los elementos que se hayan marcados cumplen las siguientes funciones:

- **Button**: Pulsador que permite enseñar al sensor Pixy un objeto. Posicionando el objeto deseado delante del sensor y manteniendo presionado el pulsador se consigue que el sensor almacene la información y pueda detectar el color del objeto.
- **USB**: Puerto USB de patilla Mini-B que permite alimentar el sensor y conectar con un ordenador.
- **I/O port**: Puerto serial que permite la comunicación del sensor Pixy con microcontroladores.
- **RC servo**: Puerto que conecta los servomotores al sensor Pixy.
- **Power in**: Conexión que permite suministrar potencia eléctrica.

3.1.2 Características de los servomotores

El montaje contará con dos servomotores GS-9018 que contará con las características siguientes [4]:

- **Sistema de control**: Modulación de ancho de pulso positiva, punto neutro a 1.5 ms
- **Rango de tensión de operación**: [4.8, 6.0] V.
- **Rango de temperatura de operación**: [-20, 60] ºC
- **Peso**: 9 gramos.
- **Par aplicado y velocidad de giro**:

	4.8 V	**6.0 V**
Par aplicado [Nm]	0.1471	0.1667
Velocidad de giro sin carga [RPM]	76.92	83.33

Tabla 1. Par aplicado y velocidad de giro del servomotor GS-9018

En la figura siguiente puede observarse el servomotor GS-9018.

Figura 3-3. Servomotor GS-9018

3.2 Montaje del sistema pan/tilt

En esta sección se presentará de forma resumida la guía de montaje seguida, obtenida de [5], así como el material empleado y las herramientas necesarias.

3.2.1 Herramientas necesarias

Para realizar el montaje ha sido necesario:

- Destornillador con punta de estrella.
- Cortador de alambre.

3.2.2 Material empleado

En la tabla siguiente se recoge el material empleado y las cantidades de cada material:

Material	Cantidad	Imagen
Tornillo 4-40 largo	6	
Tornillo 4-40 corto	4	
Tornillo corto	2	
Tornillo para servo	2	
Brazo de servo de nylon	1	
Brazo de servo circular de nylon	1	
Soporte acrílico	1	

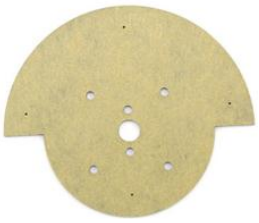
Base acrílica	1	
Brida	4	
Escuadra metálica de soporte con dos roscas	5	
Casquillo de nylon	2	
Barra metálica	1	
Pie de goma	4	

Tabla 2. Materiales para el montaje

3.2.3 Guía de montaje

En esta subsección se presentará una guía sencilla y resumida del montaje ilustrada con imágenes:

1. Cortar el brazo del servo por el quinto agujero y unirlo al servomotor con un tornillo de servo de forma que el recorrido de la carrera a contrarreloj resulte tal y como se muestra a continuación:

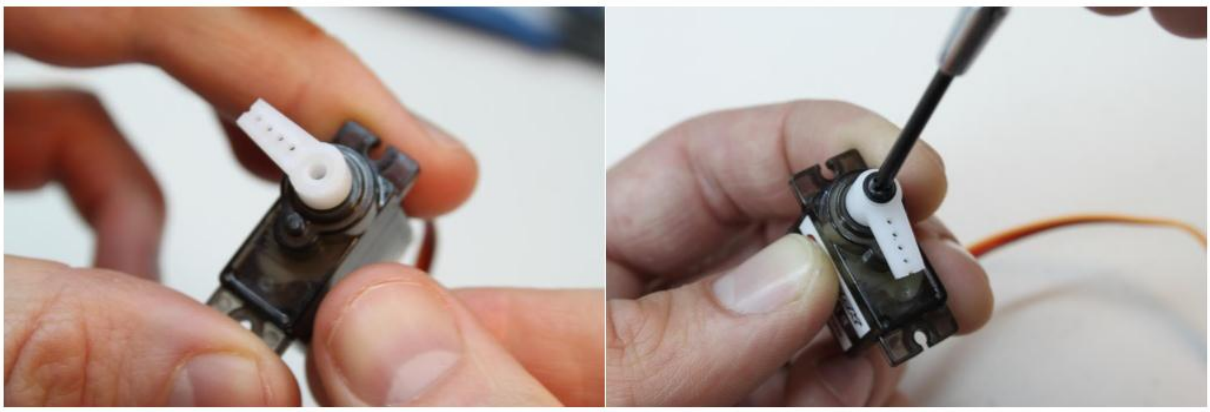


Figura 3-4. Montaje: paso 1

2. Montar el servomotor del paso anterior en el soporte acrílico y fijarlo con dos bridas tal y como se muestra a continuación:

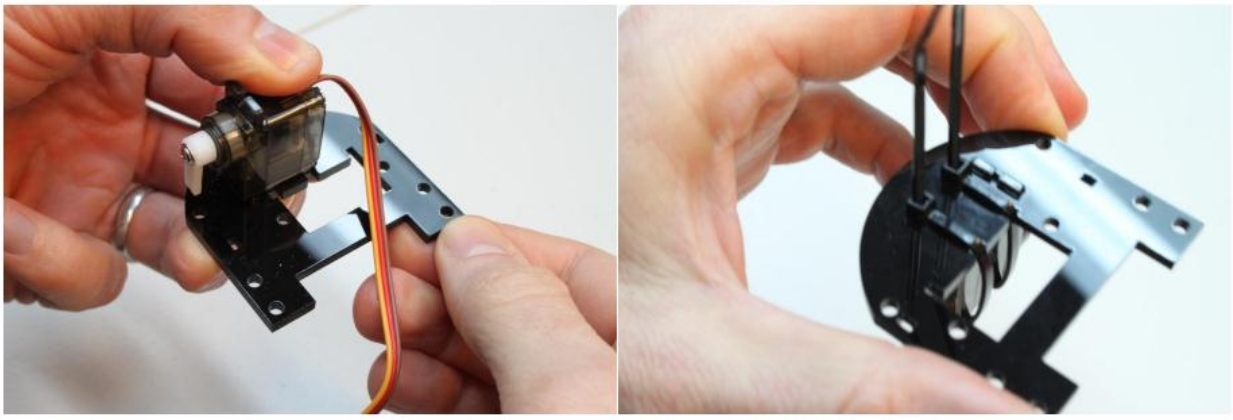


Figura 3-5. Montaje: paso 2

3. Montar el segundo servomotor en el soporte acrílico con el servo apuntando hacia abajo y fijarlo al soporte con dos tornillos 4-40 largos tal y como se muestra a continuación:

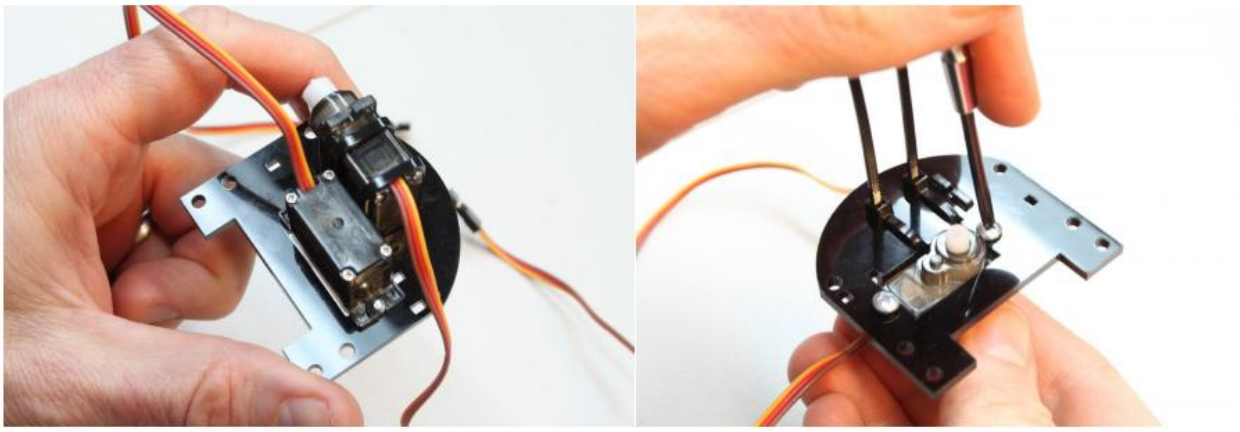


Figura 3-6. Montaje: paso 3

4. Fijar dos escuadras de soporte al soporte acrílico usando dos tornillos 4-40 largos tal y como se muestra a continuación:

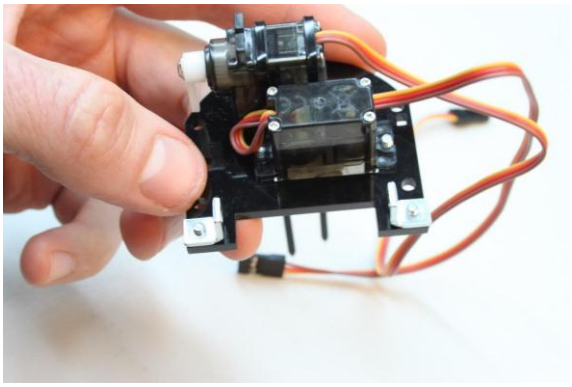


Figura 3-7. Montaje: paso 4

5. Fijar dos escuadras de soporte al sensor Pixy usando dos tornillos 4-40 cortos tal y como se muestra a continuación:

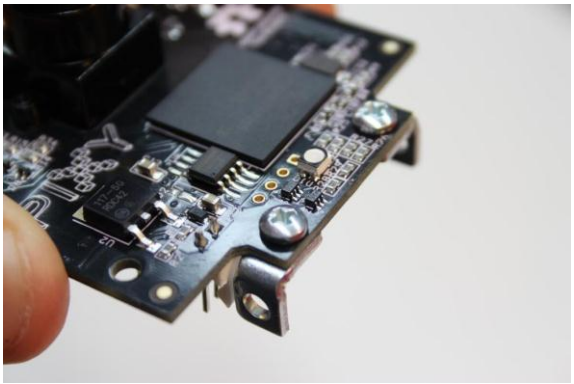


Figura 3-8. Montaje: paso 5

6. Introducir los casquillos de nylon en los orificios no atornillados de las escuadras unidas al sensor Pixy tal y como se muestra a continuación:

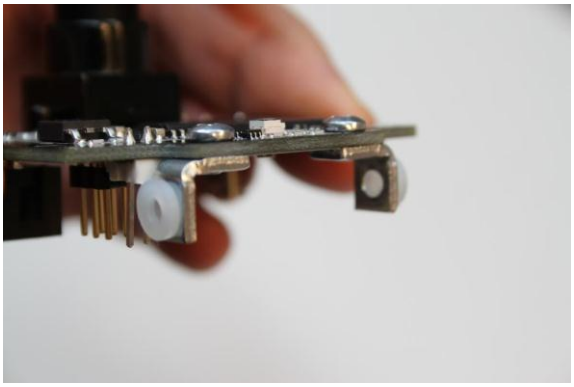


Figura 3-9. Montaje: paso 6

7. Unir el sensor Pixy al soporte acrílico empleando dos tornillos 4-40 largos tal y como se muestra a continuación:

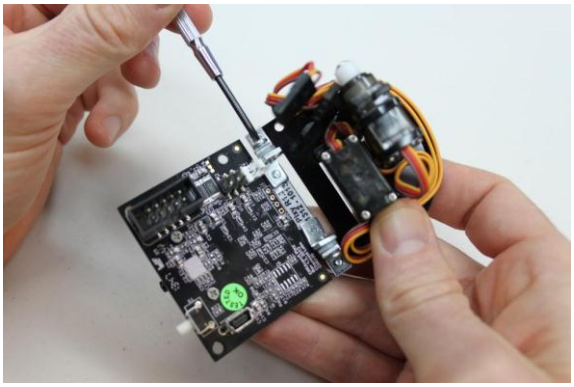


Figura 3-10. Montaje: paso 7

8. Conectar los cables de los servos al sensor Pixy y ajustarlo usando bridas tal y como se muestra a continuación:

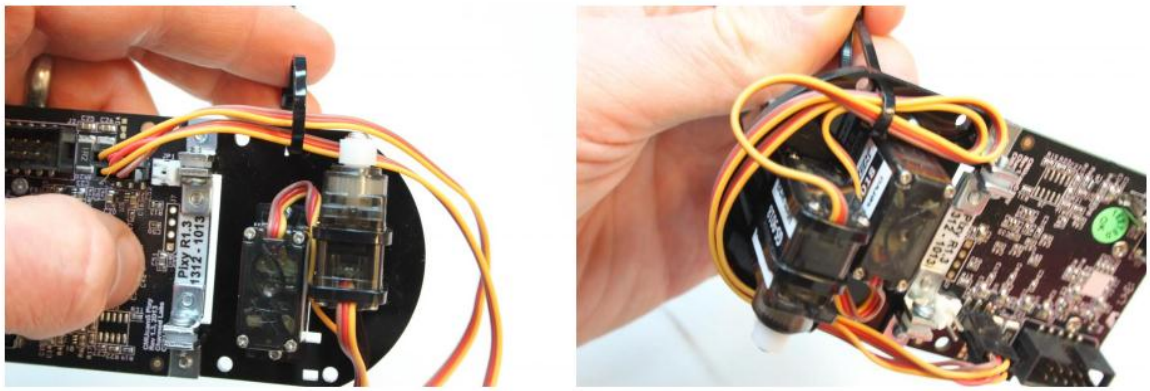


Figura 3-11. Montaje: paso 8

9. Unir el brazo de servo circular a la base acrílica usando dos tornillos cortos tal y como se muestra a continuación:

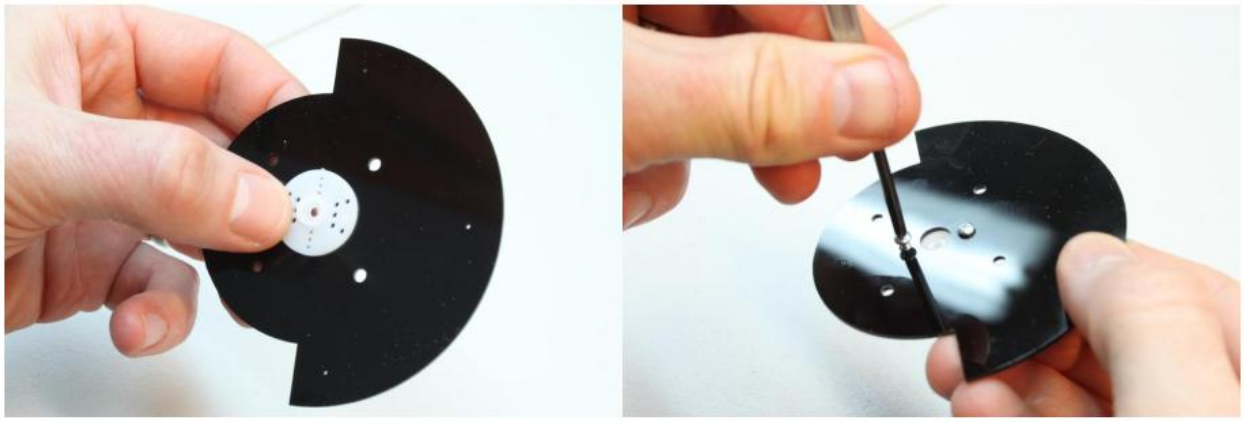


Figura 3-12. Montaje: paso 9

10. Colocar los pies de goma en la base acrílica:

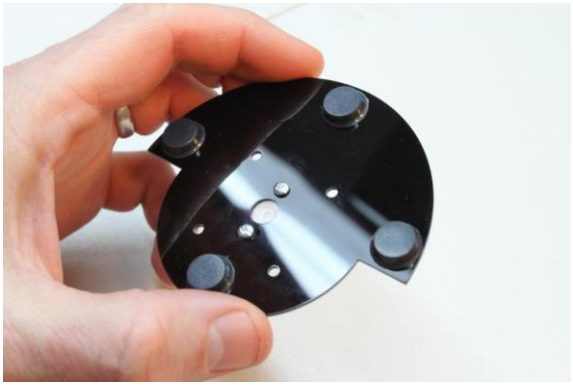


Figura 3-13. Montaje: paso 10

11. Unir el montaje anterior a la base acrílica conectando el segundo servomotor al brazo de servo circular y fijándolo con un tornillo de servo tal y como se muestra a continuación:

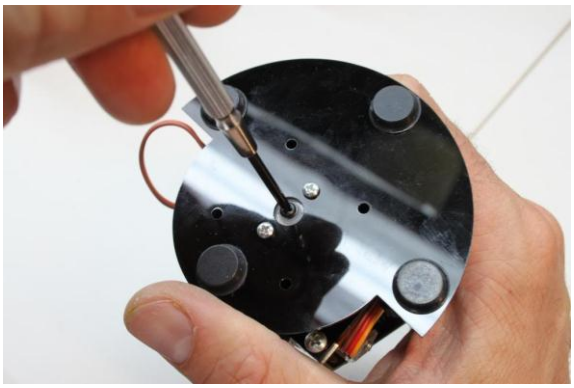


Figura 3-14. Montaje: paso 11

12. Unir la varilla metálica por el extremo cerrado a una escuadra de soporte usando un tornillo 4-40 corto tal y como se muestra a continuación:

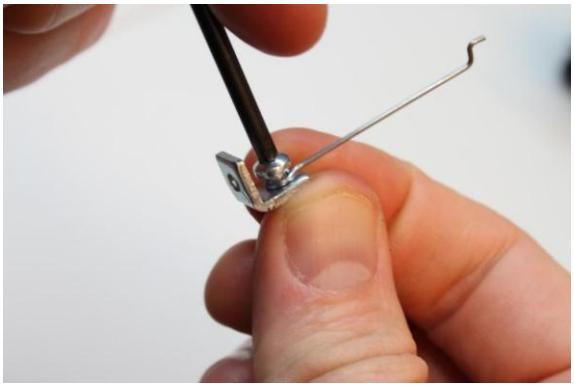


Figura 3-15. Montaje: paso 12

13. Introducir la varilla por el extremo libre en el brazo del primer servomotor y fijar la escuadra de soporte del otro extremo en el sensor Pixy tal y como se muestra a continuación:

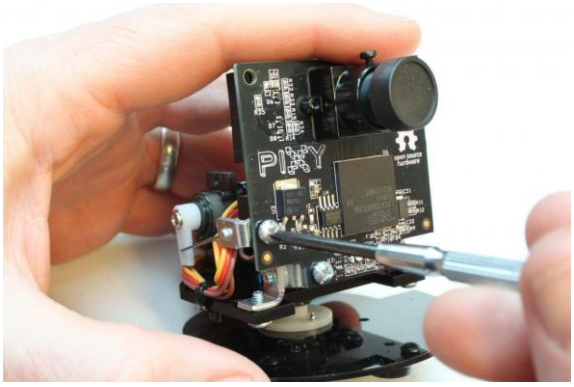


Figura 3-16. Montaje: paso 13

3.3 Aplicación para ordenador PixyMon

El sensor Pixy cuenta con una aplicación de escritorio llamada PixyMon que permite interactuar directamente con el sensor Pixy únicamente conectándolo a un ordenador a través de una conexión USB. Esta aplicación permite ver lo que el sensor Pixy capta, ejecutar programas y modificar parámetros de configuración de Pixy. Esta aplicación puede descargarse desde [6].

3.3.1 Interfaz principal

Una vez instalada la aplicación se puede observar la interfaz siguiente:

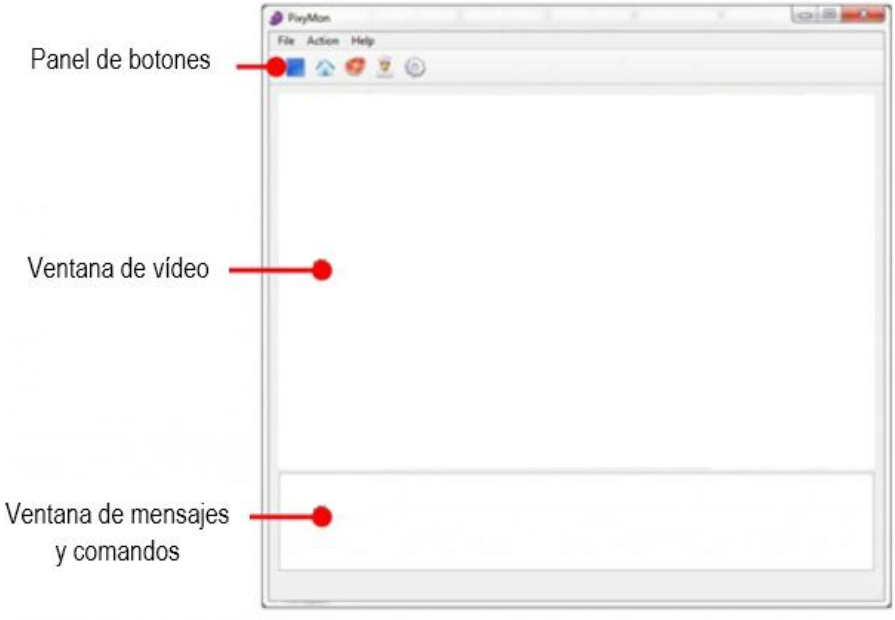


Figura 3-17. Interfaz de PixyMon

Donde se distinguen los elementos siguientes:

- **Panel de botones**: barra de herramientas más comunes.
- **Ventana de vídeo**: ventana donde se muestra el vídeo captado por el sensor Pixy
- **Ventana de mensajes y comandos**: ventana desde la que Pixy notifica ciertos mensajes y desde donde se pueden introducir comandos.

El panel de botones cuenta con los elementos siguientes:

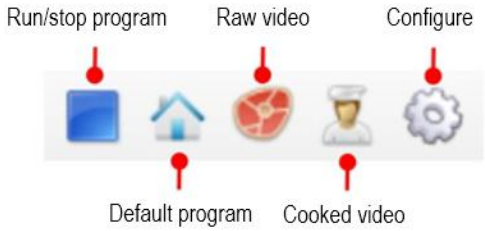


Figura 3-18. Panel de botones

Donde cada botón realiza las funciones siguientes:

- **Run/stop program**: este botón permite detener la transmisión de vídeo a través de la ventana de vídeo y de detener o reanudar el programa que se esté ejecutando.
- **Default program**: ejecuta el programa que se encuentra cargado.
- **Raw video**: muestra el vídeo sin procesar. Resulta útil para ajustar el enfoque y la iluminación.
- **Cooked video**: muestra el vídeo sin procesar con una capa de vídeo procesado. En este modo no se envía información a través de los puertos seriales.
- **Configure**: se abre el menú de configuración de parámetros.

3.3.2 Enseñar un objeto a Pixy

La aplicación PixyMon permite guardar en la memoria hasta 7 señales de colores distintas. Para ello es necesario conectar el sensor Pixy a un ordenador usando conexión USB e instalar la aplicación en el ordenador. Una vez instalada es posible almacenar colores para que el sensor los detecte siguiendo el procedimiento siguiente:

1. Una vez abierta la interfaz principal de PixyMon, activar la opción **Raw video** y mantener el objeto que se quiere enseñar a Pixy y seleccionar la opción **Set signature 1** dentro de la opción **Action** de la barra de herramientas, tal y como se muestra en la figura siguiente:

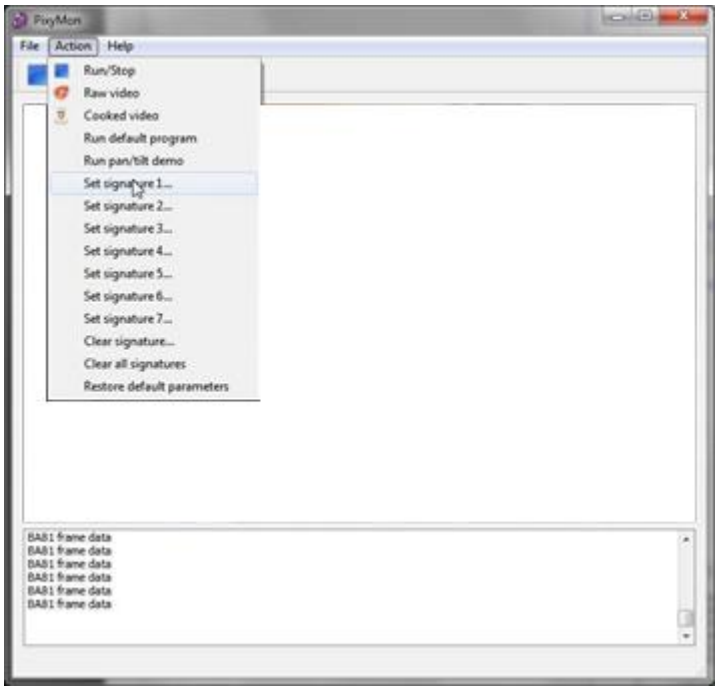


Figura 3-19. Enseñar un objeto: paso 1

2. Una vez seleccionada la señal en la que se desea guardar la información el vídeo parará de refrescarse y podremos seleccionar la región a analizar. Para ello, pulsar y mantener el botón izquierdo del ratón y arrastrar hasta abarcar el área deseada, tal y como se muestra en la figura siguiente:

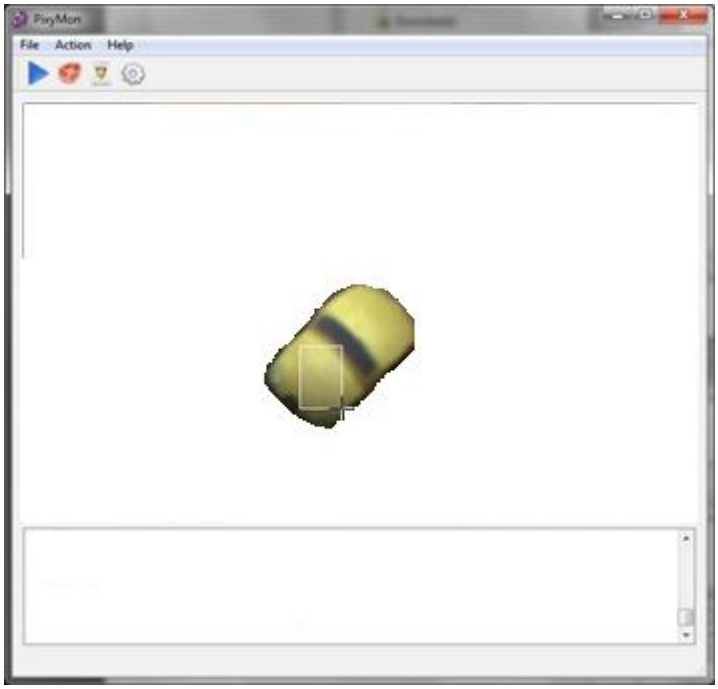


Figura 3-20. Enseñar un objeto: paso 2

3. Una vez realizado, pulsar el botón **Cooked video** del panel de botones para observar cómo el sensor Pixy detecta los píxeles del color guardado. En la figura siguiente puede observarse el resultado:

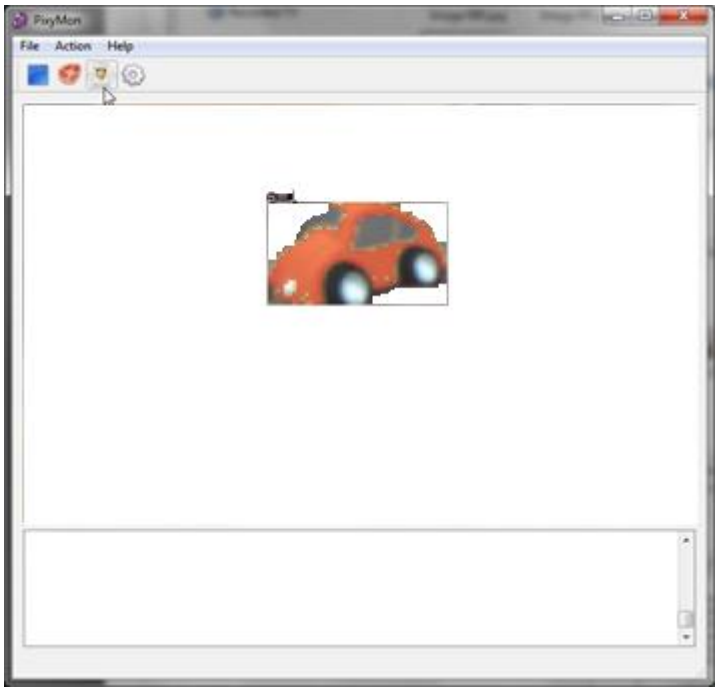


Figura 3-21. Enseñar un objeto: paso 3

Es necesario remarcar que las señales guardadas se almacenan en una memoria flash, por lo tanto, al cesar el suministro de energía eléctrica la información de las señales se perderá.

3.3.3 Mejorar la detección de objetos

Es posible mejorar la detección de las señales simplemente consiguiendo una buena iluminación del objeto, evitando sobreexposiciones o baja cantidad de iluminación. Así mismo, es posible mejorar la detección eliminando del entorno posibles objetos con colores parecidos al que se desea enseñar o empleando objetos con colores muy definidos.

Al margen de las técnicas de preparación, es posible mejorar la calidad de la detección haciendo uso de ciertas funcionalidades de la aplicación PixyMon que se desarrollan a continuación.

- **Calibrado de la señal**

El objetivo de este método es modificar el rango de inclusión de cada señal para determinar el límite según el cual se acepta o no cada pixel como señal. Para modificar este grado de inclusión sería necesario entrar en **configuración** en la opción **Signature Tuning**. Una vez en esta opción, es posible modificar el umbral de inclusión de todas las señales por separado tal y como se puede observar en la figura siguiente:

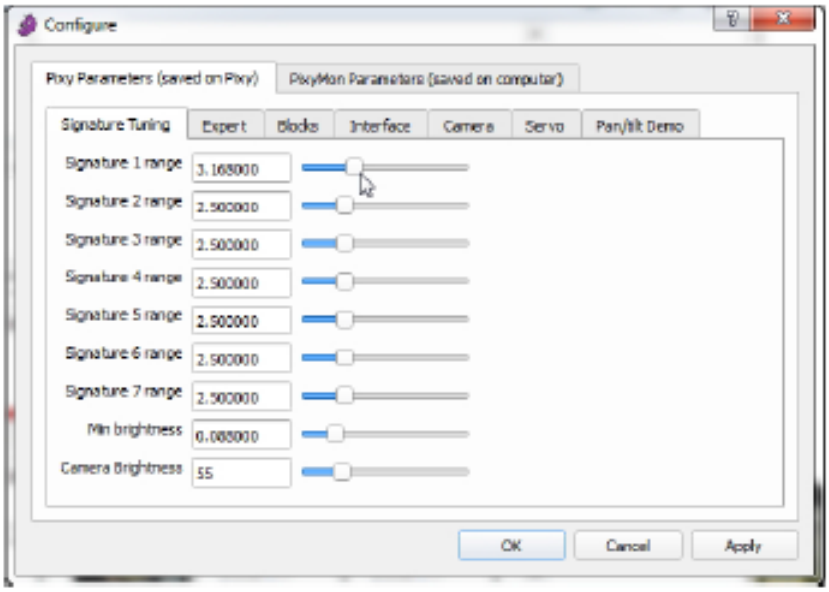


Figura 3-22. Calibrado de la señal

Cuanto menor sea el valor del umbral menos restrictiva es la inclusión, dando lugar a posibles falsos positivos. Por el contrario, cuanto mayor sea el valor del umbral mayor es la posibilidad de obtener falsos negativos. En las figuras siguientes podemos observar estos dos casos extremos:

Figura 3-23. Calibrado de la señal: falsos positivos

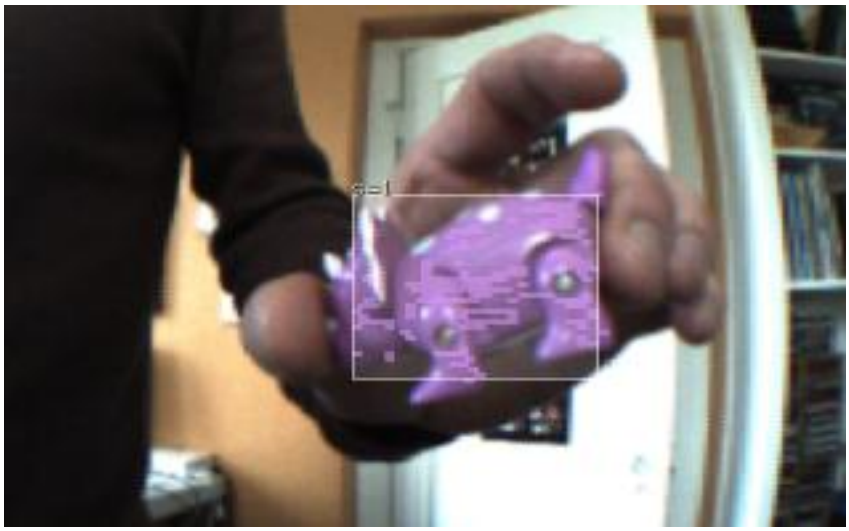


Figura 3-24. Calibrado de la señal: falsos negativos

- **Resaltado de sobreexposición y ajuste del brillo de la cámara**

Esta opción muestra de color negro las regiones de la imagen que se encuentran sobreexpuestas a iluminación. Para activar esta opción entramos en la configuración, dentro de los parámetros de PixyMon guardados en el ordenador y activamos la opción **Highlight overexposure**.

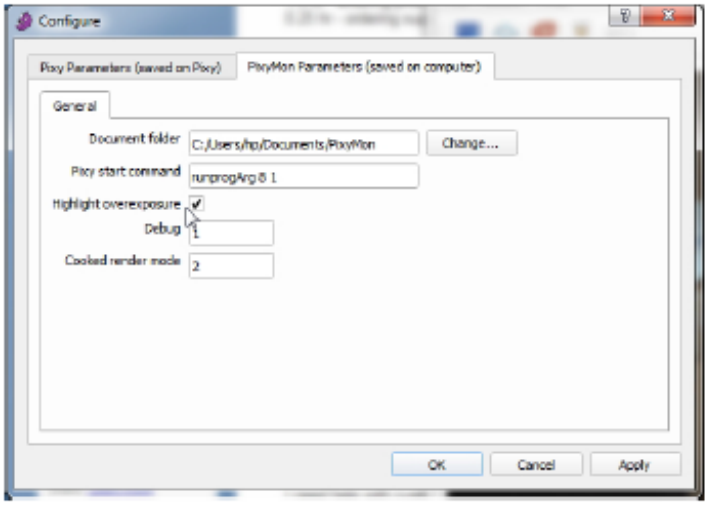


Figura 3-25. Activación del resaltado de sobreexposición

Figura 3-26. Imagen sobreexpuesta

Para reducir estas áreas sobreexpuestas es posible ajustar el brillo de la cámara desde **Signature Tuning** en la opción **Camera Brightness** para reducir las zonas afectadas.

Figura 3-27. Brillo corregido

- **Ajuste del brillo mínimo**

Desde la pestaña **Signature Tuning** es posible modificar el brillo mínimo necesario para que un pixel sea considerado como una señal. Para evitar falsos negativos es posible aumentar este umbral.

- **Ajuste del umbral de aprendizaje la señal**

Desde la pestaña **Expert** de configuración es posible modificar el umbral de inclusión que el sensor Pixy emplea cuando está aprendiendo un color determinado. Por lo que, cuanto mayor es el umbral, más elementos considera iguales a la región de aprendizaje marcada. Por el contrario, cuanto menor es el umbral, menos elementos considera iguales a la región de aprendizaje.

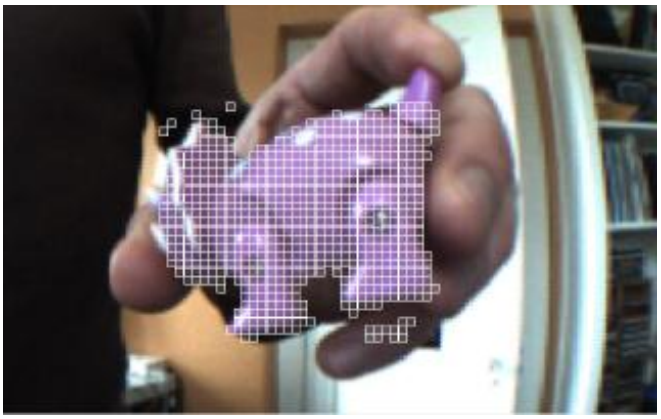


Figura 3-28. Umbral de aprendizaje demasiado alto

Figura 3-29. Umbral de aprendizaje demasiado bajo

3.3.4 Ejecutar el pan/tilt demo

La aplicación PixyMon dispone de un programa guardado en la memoria que mueve los servomotores para perseguir los objetos que coincidan con la señal 1. Este programa consiste en una aproximación de controlador PD (aproximado porque no contempla el tiempo, sino únicamente las variaciones en la posición entre iteraciones) que busca ubicar el centro del objeto detectado en el centro de la imagen.

Para ejecutar el programa pulsamos la opción **Run the pan/tilt demo** dentro de **Action** en la barra de herramientas.

3.3.5 Configuración de Pixy en PixyMon

En esta subsección se resumirán los distintos menús de configuración de Pixy en la aplicación PixyMon. La configuración cuenta con 8 menús de configuración distintos dentro de la opción **Pixy Parameters**.

- **Signature Tuning**

  Tal y como se desarrolló en las subsecciones anteriores, esté menú de configuración permite modificar el umbral de inclusión de las señales almacenadas en memoria, modificar el brillo mínimo aceptable y el brillo de la cámara

- **Signature Labels**

  Este menú de configuración permite modificar el nombre que muestran las señales almacenadas en el vídeo al ser detectadas.

- **Expert**

  Este menú de configuración contiene parámetros avanzados del sensor Pixy. Se recomienda no modificarlos.

- **Blocks**

  Cada objeto que el sensor Pixy detecta es catalogado como "block" que lleva asociado una serie de parámetros que permiten identificar a qué señal corresponde, su tamaño y ubicación en la imagen. Este menú de configuración permite modificar el número máximo de "blocks" que se permiten, el máximo número de "blocks" por cada señal el área mínima (ancho·alto) a partir del cual se acepta la detección como un "block"

- **Interface**

  Este menú de configuración contiene los parámetros de comunicación de Pixy hacia el exterior. Es posible seleccionar el puerto de salida de datos (Arduino IC SP SPI, I2C, UART,…), la dirección del puerto de salida I2C y la tasa de baudios del Transmisor-Receptor Asíncrono Universal (UART).

- **Camera**

  Este menú de configuración contiene parámetros acerca de cómo el sensor Pixy adquiere imágenes para su procesado. Se recomienda no modificarlos.

- **Servo**

  Este menú de configuración contiene parámetros de configuración de los puertos en el sensor Pixy de los dos servomotores. Los límites superior e inferior representan como de lejos se puede mover el servo en sentido antihorario y en sentido horario respectivamente.

- **Pan/tilt Demo**

  Este menú de configuración contiene los parámetros que permiten ajustar el controlador de persecución. Es posible modificar las ganancias P y D asociadas al giro vertical y horizontal del controlador

4. ARDUINO

Arduino es una plataforma de software libre basada en la filosofía del uso sencillo del hardware y del software. Consiste en una placa con circuitería cuyo microcontrolador puede ser programado y en una plataforma de software llamada Arduino IDE (Entorno de desarrollo integrado) que permite cargar códigos desarrollados desde un ordenador.

Arduino cuenta con las siguientes características globales:

- La placa de Arduino es capaz de leer señales analógicas y digitales de diferentes sensores y convertirlas en una salida concreta como activar un motor o encender un LED.
- Es posible controlar las funciones de la placa de Arduino enviando un conjunto de instrucciones al microcontrolador empleando Arduino IDE.
- Para cargar un programa en el microcontrolador únicamente es necesario un cable USB y la aplicación de Arduino para ordenador.
- El lenguaje empleado por Arduino IDE es una versión simplificada de C++, lo cual simplifica mucho el aprendizaje del lenguaje.
- Es posible programar funciones en Arduino IDE sin necesidad de trabajar directamente con las librerías.

Existen multitud de tipos de placas de Arduino disponibles en las que puede variar el número de entradas o salidas, la velocidad, la tensión operativa, etc. La placa de Arduino empleada para este proyecto es Arduino UNO, ya que es la de uso más extendido y es la recomendada para iniciarse en la programación con Arduino.

4.1 Descripción de la placa de Arduino

En esta sección se presentarán los elementos que componen la placa de Arduino y se remarcarán aquéllas que tomarán parte directa en el desarrollo del proyecto. En la figura siguiente puede observarse la placa de Arduino UNO con sus elementos enumerados:

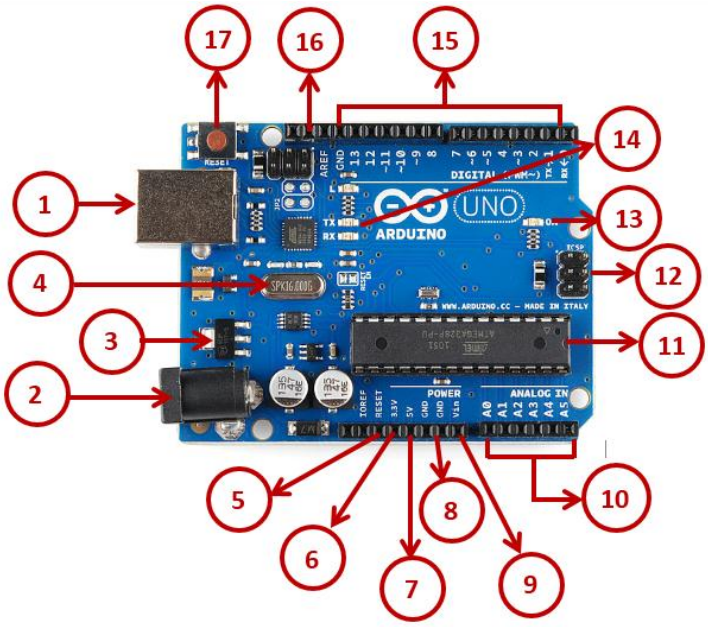


Figura 4-1. Placa de Arduino UNO

Donde:

1. **Alimentación USB**: Arduino puede alimentarse conectándose a un ordenador empleando un cable USB tipo B.
2. **Alimentación**: Arduino puede alimentarse directamente de la red eléctrica empleando un alimentador.
3. **Regulador de tensión**: Controla la tensión entregada a la placa de Arduino y estabiliza las tensiones en continua entregadas al procesador y otros elementos.
4. **Cristal oscilador**: Se emplea para calcular el tiempo.
5. **Pin de reinicio de Arduino**: Permite conectar un botón de reseteo externo a la placa de Arduino.
6. **Pin de Arduino**: Pin que suministra 3.3 V de salida.
7. **Pin de Arduino**: Pin que suministra 5 V de salida.
8. **Pin de Arduino**: Pin empleado como tierra para los circuitos.
9. **Pin de Arduino**: Pin empleado para alimentar la placa de Arduino con una fuente externa (como corriente alterna).
10. **Pines analógicos**: 5 pines que permiten leer señales analógicas de sensores externos y convertirlas en señales digitales.
11. **Microcontrolador principal**.
12. **Pin ICSP**: pin que permite conectar un periférico. Comúnmente referido como SPI.
13. **Indicador LED de potencia**: LED que se ilumina al conectar la placa de Arduino a la corriente adecuada.
14. **LEDs TX y RX**: LEDs que se iluminan según se transmita o se reciba respectivamente.
15. **Pines digitales I/O**: la placa de Arduino UNO posee 14 pines digitales I/O, de los cuales 6 proporcionan salida PWM (Ancho de Pulso Modulado).
16. **AREF**: Se emplea para fijar una referencia externa de tensión.
17. **Botón de reinicio de Arduino**: Botón que permite reiniciar la placa de Arduino.

Para el desarrollo del proyecto será necesario emplear únicamente dos de los elementos anteriores:

- **Alimentación USB (1)**: a través de este puerto se alimentará la placa de Arduino UNO, se cargarán los programas y se recibirá información desde la placa para su posterior procesado.
- **Pin ICSP (12)**: a través de este pin se realizará la conexión entre la placa Arduino UNO y el sensor Pixy (tratándose este como un elemento periférico).

4.2 Arduino IDE

Para poder trabajar con Arduino IDE es necesario descargar e instalar la aplicación de escritorio de Arduino que puede ser descargada desde [7].

Una vez instalada e iniciada la aplicación de escritorio, y tras conectar la placa de Arduino al ordenador, es necesario seleccionar la placa de Arduino para evitar errores durante la carga del programa a la placa. Para ello accedemos al menú **Herramientas** de la barra de herramientas, y seleccionamos Arduino UNO en la opción **Placa**, tal y como puede observarse en la figura siguiente:

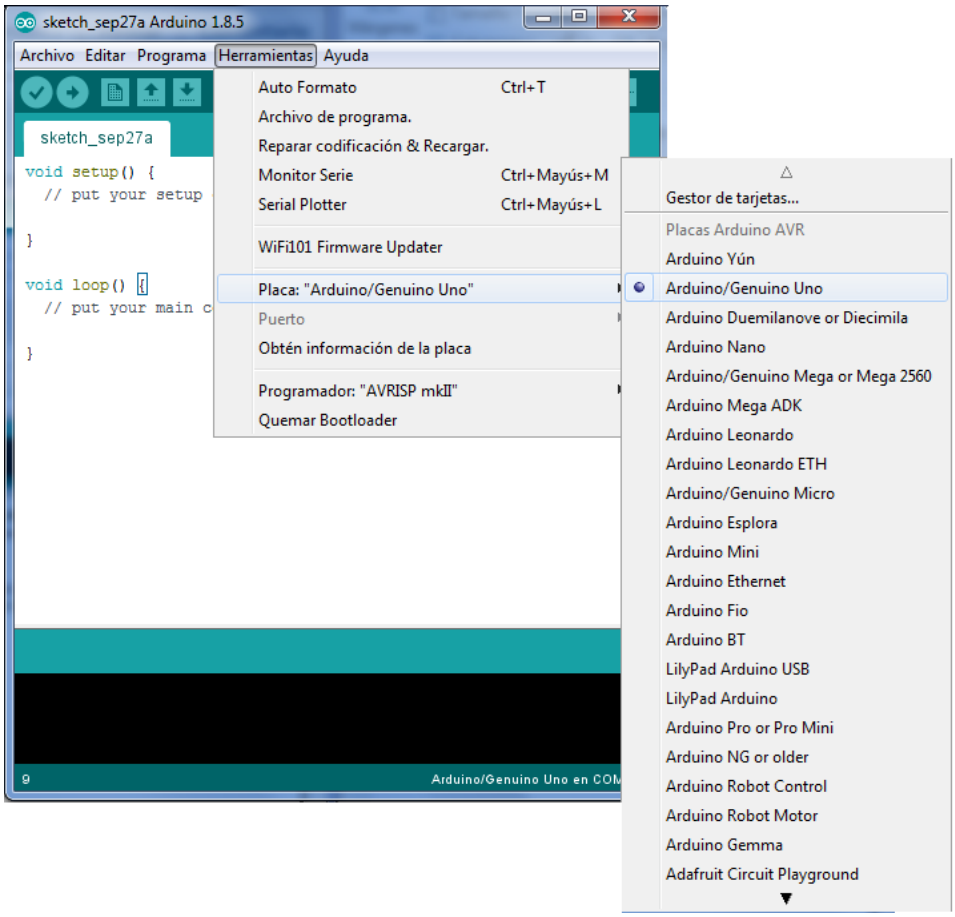


Figura 4-2. Selección de placa de Arduino

Previamente a la presentación de la estructura del lenguaje de Arduino y de los elementos relativos al código empleados en este proyecto se hará mención de algunas funcionalidades de interés de la barra de herramientas de la interfaz. En la figura siguiente puede observarse esta barra de herramientas.

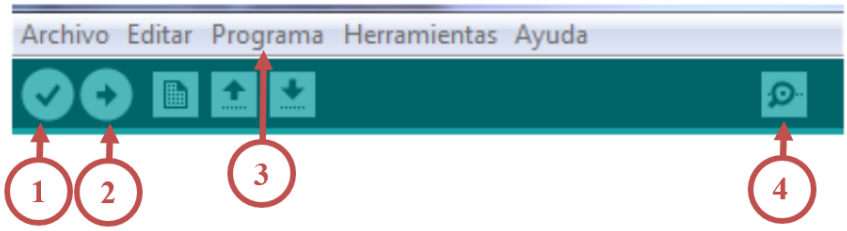


Figura 4-3. Barra de herramientas de Arduino

Donde:

1. Ejecuta un corrector que permite verificar si existe algún error en el código.
2. Permite cargar el programa desarrollado en la placa de Arduino.
3. Contiene ciertas funciones relacionada con el código, entre ellas la opción de incluir librerías en el paquete de librerías de Arduino. Esto será de utilidad puesto que el sensor Pixy cuenta con su propia librería .zip que será necesario cargar previamente en la aplicación de Arduino.
4. Despliega una ventana donde se muestra la información enviada por el puerto serial una vez se ejecuta el programa en caso de que se hubiese escrito la orden en el código cargado.

4.2.1 Estructura de Arduino IDE

La estructura básica de un programa de Arduino consta de dos partes en su forma más simple. Estas dos partes son indispensables para su ejecución y en ellas contienen las instrucciones y declaraciones. Estas dos partes se conocen como **setup** y **loop**.

En **setup** se recoge la configuración del programa y es la primera función que se ejecuta en el programa. Esta función sólo se ejecuta una única vez y debe incluirse aunque en ella no se declare nada. En este bloque pueden declararse variables, inicial las comunicaciones seriales, etc.

En **loop** se encuentra contenido el grueso del programa a ejecutar. Tal y como su nombre indica, se trata de un bucle que se repite cíclicamente y sin pausa.

Dos elementos fundamentales del lenguaje de Arduino son las llaves {} y el punto y coma ;. Las llaves sirven para definir el principio y el final de un bloque de instrucciones, empleándose en funciones, bucles, condiciones, etc. El punto y coma se emplea para separar instrucciones y debe emplearse al finalizar cada instrucción.

```
void setup()
{
  instrucciones;
}

void loop()
{
  instrucciones;
}
```

4.2.2 Elementos de Arduino IDE empleados

- **Funciones**

Una función es un bloque de código que se ejecuta cuando se llama a la función. Las funciones se declaran según el tipo de valor "**type**" que la función devuelva. Si la función devuelve un valor entero se iniciarán con **int**¸ mientras que si no se devuelve ningún valor, se iniciará con **void**.

Para construir una función bastaría con iniciar con el tipo de valor, seguido del nombre y acabando con los parámetros de entrada entre paréntesis.

```
type nombre(parámetros)
{
  instrucciones;
}
```

- **Variables**

En arduino las variables deben declararse previo a su uso y de acuerdo al tipo de información que almacenarán.

```
type nombre;
```

Las variables cuentan con otra propiedad añadida, en función del lugar en el que se declare la variable, ésta podrá ser o no visible para ciertas partes del programa. Una variable **global** es aquella que puede ser vista por cualquier función e instrucción del programa y para ello debe declararse antes del **setup**, al comienzo del programa. Una variable **local** es aquella que se define dentro de un bloque de instrucciones y sólo es visible para ese bloque concreto y para ello debe declararse dentro de ese bloque.

En la tabla siguiente se recogen los tipos utilizados con la clase de información que almacenan.

type	Tipo de datos
int	Número entero de 16 bits
int32_t	Número entero de 32 bits
double	Punto flotante de doble precisión en 32 bits
boolean	Valor binario: 0/1 o true/false

Tabla 3. Tipo de variables empleadas

- **Array**

Un array es un vector al que se accede a cada componente con un número de índice, estando el primer valor bajo el índice 0. Un array se declara escribiendo el tipo de datos que va a almacenar, el nombre, el número de componentes que tendrá entre corchetes, un símbolo = y finalizando con los valores entre llaves y separados por comas.

Type *nombre*[tamaño] = {valor0, valor1, …, valorN};

- **Aritmética**

Los operadores que se incluyen son únicamente suma, resta, multiplicación y división.

```
y = y + 3;
x = x - 7;
i = j * 6;
r = r / 5;
```

Además, se disponen de unas asignaciones compuestas que simplifican algunas operaciones. Estas asignaciones se recogen en la tabla siguiente.

Asignación	Equivalencia
$x++$	$x = x + 1$
$x--$	$x = x - 1$
$x += y$	$x = x + y$
$x -= y$	$x = x - y$
$x *= y$	$x = x * y$
$x /= y$	$x = x/y$

Tabla 4. Asignaciones compuestas de Arduino IDE

- **pow**

Función de Arduino IDE que permite calcular una base elevado a un exponente.

pow(base, exponente);

- **if… else**

Bloque de instrucciones en el que se ejecuta la primera parte si se cumple una condición o la se ejecuta la segunda si no se cumple la condición.

```
if (condición)
{
    instrucciones 1;
}
else
{
    instrucciones 2;
}
```

- **for**

Bloque de instrucciones que permite repetir las instrucciones en un bucle. Este bucle se repite desde un valor inicial hasta otro final, variando este valor según una expresión concreta.

```
for (int j = 0; j<10; j++)
{
  instrucciones;
}
```

- **delay**

Detiene la ejecución del programa una cantidad de tiempo en milisegundos indicada en la instrucción.

```
delay(1000);
```

- **millis**

Devuelve el número de milisegundos transcurrido desde el inicio del programa hasta el momento actual.

```
valor = millis();
```

- **Serial.begin()**

Abre el puerto serie y fija la velocidad en baudios para la transmisión de datos en serie. El valor típico de la velocidad es 9600.

```
void setup()
{
Serial.begin(9600);
}
```

- **Serial.print()**

Imprime los datos en el puerto serie.

```
Serial.print();
```

4.3 Pixy y Arduino

Para trabajar con el sensor Pixy es necesario hacer uso de las librerías desarrolladas por los creadores de Pixy para poder crear cualquier programa que deseemos que se ejecute. Las librerías se encuentran en un archivo comprimido descargable desde [6]. Tal y como se vio anteriormente, es posible añadir librerías desde la interfaz principal de la aplicación de Arduino para ordenador.

Una vez añadida la librería, es necesario escribir las siguientes instrucciones al comienzo del programa que vaya a cargarse en el sensor Pixy.

```
#include <SPI.h>
#include <Pixy.h>
Pixy Pixy;
```

Las dos primeras líneas incluyen la librería SPI propia de Arduino IDE y una librería propia de los desarrolladores de Pixy. La tercera línea es la declaración de un objeto llamado Pixy. Estas librerías ponen a disposición del programador una serie de funciones específicas del sensor Pixy que permiten modificar la posición de los servomotores y obtener información sobre los "blocks".

Las funciones específicas de Pixy que se encuentran para arduino y que han sido empleadas son las que se listan a continuación:

- **getBlocks()**: retorna el número de objetos que Pixy ha detectado.
- **Pixy.blocks[i].signature**: retorna el número de la señal a la que corresponde el objeto i.
- **Pixy.blocks[i].x**: retorna la posición x del centro del objeto i.
- **Pixy.blocks[i].y**: retorna la posición x del centro del objeto i.
- **Pixy.setServos(pan, tilt)**: fija la posición de los servos. El rango en el que pueden moverse es entre 0 y 1000.

5. CONTROLADOR PID

El controlador PID es uno de los controladores más empleados en los sistemas con realimentación y han probado ser muy robustos en muchas aplicaciones industriales. Este controlador presenta una estructura muy simple, lo cual facilita su implementación. Este controlador trabaja comparando el estado de un sistema con una referencia y empleando esta información para devolver una respuesta que haga disminuir el error respecto a esta referencia.

Se considerará un proceso cuya entrada está controlada por un control. Este control tiene como entrada el error en la salida del proceso.

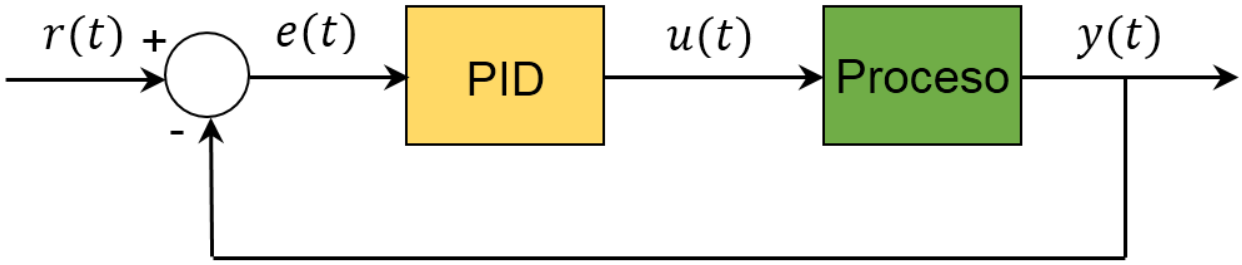


Figura 5-1. Proceso con un controlador y realimentación

El controlador PID incluye tres acciones diferentes: proporcional (P), integral (I) y derivativa (D). Donde cada acción cumple un propósito cuyas ventajas aprovecha en su conjunto el PID.

5.1 Funcionamiento y características del controlador

- **Controlador proporcional**: la salida de este controlador es proporcional al error. Un controlador proporcional puede controlar cualquier proceso estable, pero de forma limitada y genera error en régimen permanente.

$$u(t) = K_p e(t)$$

- **Controlador integral**: la salida de este controlador es proporcional al error acumulado en el tiempo transcurrido. Este controlador permite reaccionar ante perturbaciones en régimen permanente. Combinado con el controlador proporciona un régimen permanente sin error.

$$u(t) = K_i \int_0^t e(\tau) d\tau$$

- **Controlador derivativo**: la salida de este controlador es proporcional a la variación del error. Este controlador tiene carácter previsor y acelera la respuesta del controlador en periodos transitorios. Además, añade amortiguamiento al sistema. Nunca se emplea sólo, ya que amplifica el ruido.

$$u(t) = K_d \frac{de(t)}{dt}$$

- **Controlador PID**: este controlador reúne las ventajas de cada una de las acciones individuales anteriores.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt}$$

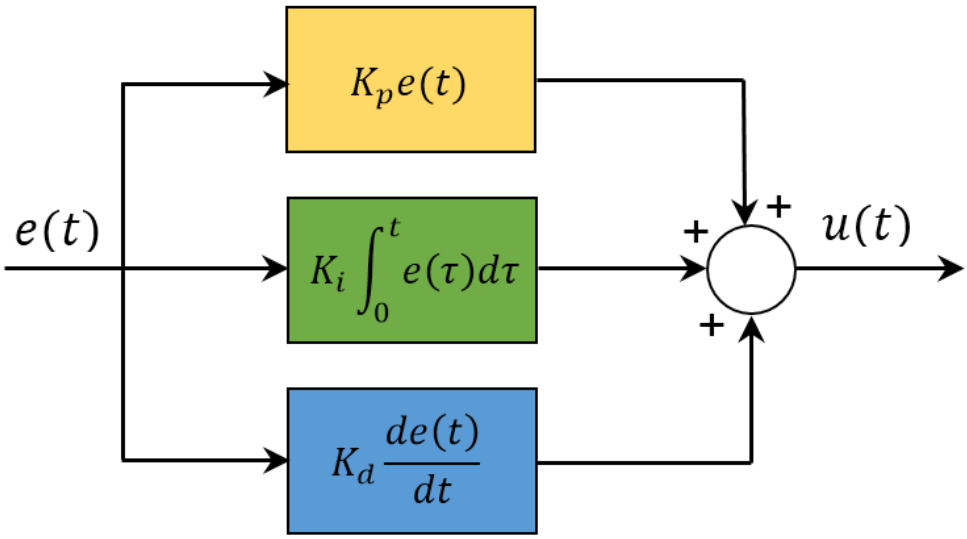


Figura 5-2. Controlador PID

5.2 Particularización

5.2.1 Proceso a controlar

Para desarrollar un algoritmo de detección de elementos será necesario el uso de un controlador PID con el que poder determinar la posición de los objetos. El proceso a controlar consistirá en centrar los objetos que sean visualizados por el sensor Pixy, es decir, mover los servomotores de forma que el objeto pase a estar en el centro del enfoque.

El sensor Pixy facilita información sobre las coordenadas de los objetos dentro de la imagen que recibe, por lo tanto, podrá compararse la posición de los objetos con el centro de la imagen. Una vez realizada esta comparación se obtendría el error y bastaría aplicar el controlador PID ajustado convenientemente para facilitar la posición a la que habría de colocar cada servomotor progresivamente hasta eliminar el error.

En la figura siguiente se observa el sistema de referencia empleado por el sensor Pixy en las imágenes que capta, el centro de un objeto detectado i y el centro del enfoque C.

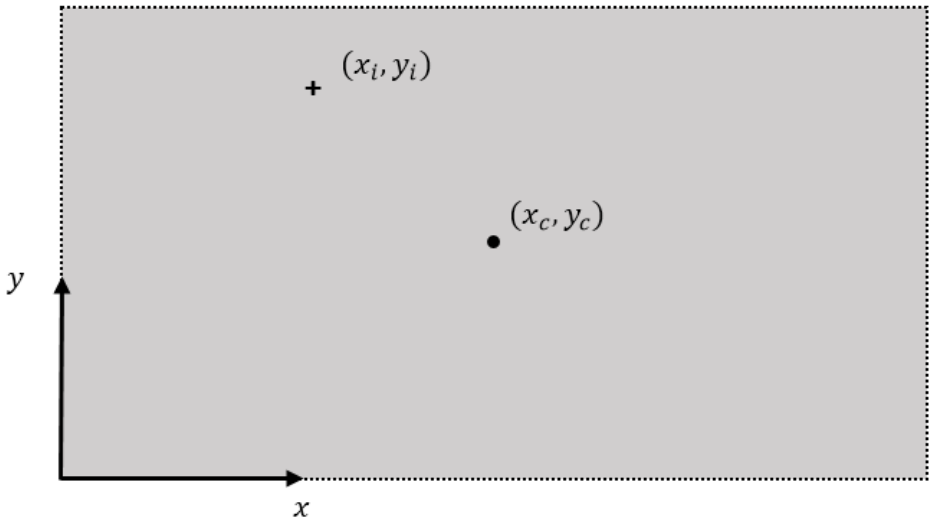


Figura 5-3. Sistema de referencia de la imagen obtenida por el sensor Pixy

La imagen se encuentra dividida de forma discreta en 320 puntos en el eje X y en 200 puntos en el eje Y.

Como se puede observar, será necesario corregir el error vertical (asociado a la coordenada y) y el error horizontal (asociado a la coordenada x). Esto se hará de forma desacoplada considerando que el error vertical será corregido modificando la posición del servomotor de movimiento vertical (tilt) y considerando que el error horizontal será corregido modificando la posición del servomotor de movimiento horizontal (pan). Por lo tanto, los errores se determinarían de la forma siguiente:

$$e_P(t) = x_C - x_i(t)$$
$$e_T(t) = y_i(t) - y_C$$

La decisión de optar por esta forma del error está basada en la usada por el controlador "PD" empleado por el Pan/tilt demo de PixyMon y se ha decidido respetarla.

Por lo tanto, se obtendría una salida asociada a cada error:

$$u_P(t) = K_{p,P}e_P(t) + K_{i,P}\int_0^t e_P(\tau)d\tau + K_{d,P}\frac{de_P(t)}{dt}$$
$$u_T(t) = K_{p,T}e_T(t) + K_{i,T}\int_0^t e_T(\tau)d\tau + K_{d,T}\frac{de_T(t)}{dt}$$

Donde la salida asociada al error horizontal $u_P(t)$ coincidirá con la posición del servomotor horizontal y la salida asociada al error vertical $u_T(t)$ coincidirá con la posición asociada al servomotor vertical. Para ello, sería necesario ajustar las constantes del controlador de forma adecuada. Dado que no está contemplado en el alcance del proyecto realizar un análisis del comportamiento dinámico del proceso, se ha optado por realizar un ajuste manual del controlador que ha resultado en los valores siguientes:

	Movimiento horizontal	**Movimiento vertical**
$\boldsymbol{K_P}$	0.1	0.1
$\boldsymbol{K_I}$	10^{-3}	10^{-3}
$\boldsymbol{K_D}$	$3 \cdot 10^{-4}$	$2 \cdot 10^{-4}$

Tabla 5. Constantes del controlador PID

A pesar de tratarse de un ajuste manual, es posible comprobar de forma experimental que el controlador PID actúa de forma rápida y sin sobre-oscilaciones.

5.2.2 Adaptación para Arduino IDE

Para implementar un controlador PID en Arduino IDE será necesario discretizar las expresiones anteriores para poder desarrollar una función que tenga como entrada el error cometido en ese instante de tiempo. Esta función almacenará el error en el instante anterior y el error acumulado hasta el momento, y empleando la función **millis** de Arduino IDE, será posible determinar unos valores aproximados de la derivada y la integral en ese instante de tiempo. Dado que tanto el controlador como el procesador de Arduino ejecutan las instrucciones a gran velocidad, estas aproximaciones serán muy aproximadas.

$$E_{a,t} = E_{a,t-\Delta t} + e_t\Delta t, \qquad E_{a,t-\Delta t}\big|_{t=0} = 0, \qquad e_{t-\Delta t}|_{t=0} = 0$$
$$u_t = K_P e_t + K_I\big(E_{a,t}\big) + K_D\frac{e_t - e_{t-\Delta t}}{\Delta t}$$

Esta discretización es que la que ha sido empleada para el desarrollo del algoritmo de rastreo y posicionamiento de elementos. Para más detalles sobre el algoritmo consultar el Apéndice B.

6. DETECCIÓN DE ELEMENTOS

En este capítulo se describirá el algoritmo de detección de elementos implementado a raíz del modelo matemático del movimiento de Pixy que se ha desarrollado. El modelo matemático del movimiento del sensor Pixy está basado en la teoría cinemática de brazos robóticos [8], que permite conocer en todo instante la posición del extremo del brazo en función de la rotación del servomotor.

El principal objetivo de este algoritmo es barrer todo el campo visual del sensor Pixy de forma discreta, implementando en cada posición de barrido el controlador PID para poder centrar cada elemento detectado y transmitirlo a través del puerto serial al ordenador para su posterior procesado en MATLAB

6.1 Modelado matemático de Pixy: movimiento vertical

El modelo matemático pretende establecer la relación entre el movimiento del servomotor del movimiento vertical y la posición de la cámara del sensor Pixy. Dado que la posición vertical del sensor está influenciada únicamente por el movimiento del servomotor vertical el modelo será bidimensional.

En el modelado del movimiento del sensor Pixy se hará uso de dos sistemas de referencia anclados a dos puntos fijos. El primer sistema de referencia se encuentra anclado al centro de giro del servomotor, mientras que el segundo sistema de referencia se encuentra anclado al punto fijo respecto al que giraría la placa del sensor Pixy. En la figura siguiente puede observarse el modelo empleado.

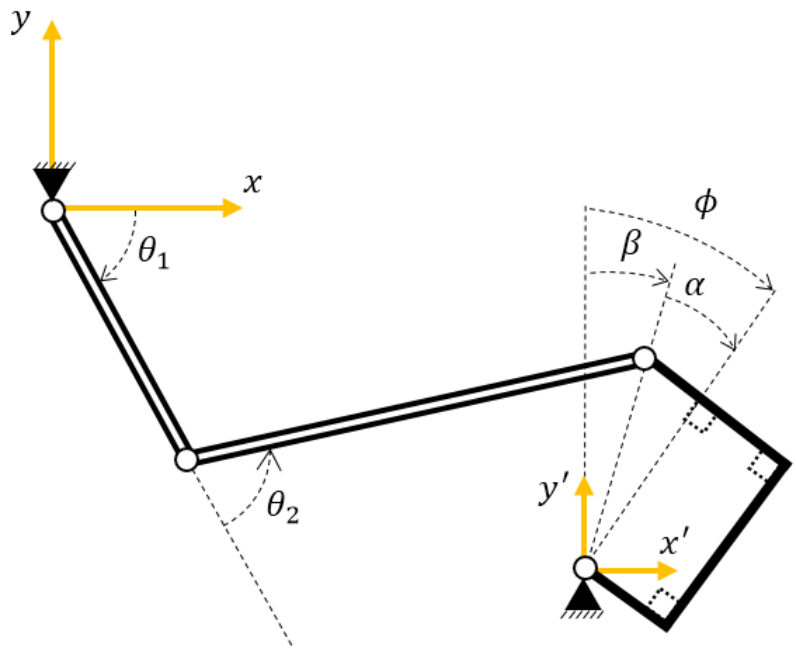


Figura 6-1. Sistemas de referencia del modelo

Donde:

- $(\boldsymbol{x}, \boldsymbol{y})$: sistema de referencia anclado al centro de giro del servomotor de movimiento vertical.
- $(\boldsymbol{x'}, \boldsymbol{y'})$: sistema de referencia anclado al punto fijo de la placa del sensor Pixy.
- $\boldsymbol{\theta_1}$: ángulo girado por el brazo del servomotor.
- $\boldsymbol{\theta_2}$: ángulo girado por la barra metálica respecto al brazo del servomotor.
- $\boldsymbol{\beta}$: ángulo formado por $\boldsymbol{y'}$ y el vector que define el extremo del brazo en el sistema $(\boldsymbol{x'}, \boldsymbol{y'})$.

- $\boldsymbol{\phi}$: ángulo formado por $\boldsymbol{y'}$ y la placa del sensor Pixy.
- $\boldsymbol{\alpha}$: diferencia entre β y ϕ.

La notación empleada para las dimensiones del modelo se puede observar en la figura siguiente:

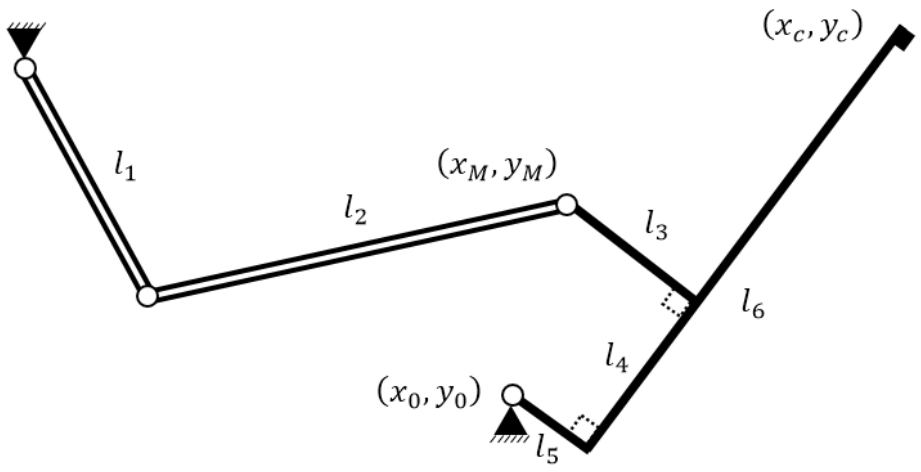


Figura 6-2. Dimensiones del modelo

Donde:

- $(\boldsymbol{x_0}, \boldsymbol{y_0})$: posición del punto fijo de la placa del sensor Pixy en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$.
- $(\boldsymbol{x_M}, \boldsymbol{y_M})$: posición del extremo del brazo en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$.
- $(\boldsymbol{x_c}, \boldsymbol{y_c})$: posición de la cámara del sensor Pixy en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$.
- $\boldsymbol{l_i}$: longitud del elemento $\boldsymbol{i}$.

Las dimensiones anteriores se recogen en la tabla siguiente:

Parámetro	Valor
$\boldsymbol{l_1}$ $[\boldsymbol{mm}]$	9.8
$\boldsymbol{l_2}$ $[\boldsymbol{mm}]$	31.7
$\boldsymbol{l_3}$ $[\boldsymbol{mm}]$	7.5
$\boldsymbol{l_4}$ $[\boldsymbol{mm}]$	12.5
$\boldsymbol{l_5}$ $[\boldsymbol{mm}]$	6
$\boldsymbol{l_6}$ $[\boldsymbol{mm}]$	38
$\boldsymbol{x_0}$ $[\boldsymbol{mm}]$	32.3
$\boldsymbol{y_0}$ $[\boldsymbol{mm}]$	-11.8
$\boldsymbol{\alpha}$ [°]	6.84

Tabla 6. Parámetros del modelo

Por lo tanto, es posible determinar la posición del extremo del brazo en función de los ángulos θ_1 y θ_2.

Si se expresa en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$ se obtendría:

$$x_M = l_1 \cos\theta_1 + l_2 \cos(\theta_2 - \theta_1)$$

$$y_M = -l_1 \sin\theta_1 + l_2 \sin(\theta_2 - \theta_1)$$

Si bien, será necesario establecer las restricciones asociadas al punto fijo entorno al que debe girar este extremo. Es decir, el extremo del brazo estaría forzado a moverse en una circunferencia alrededor de $(\boldsymbol{x_0}, \boldsymbol{y_0})$. Si se expresa esta restricción en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$ se obtendría:

$$x_M = R \sin\beta + x_0$$

$$y_M = R \cos\beta + y_0$$

Donde:

$$R = \sqrt{l_4^2 + (l_3 - l_5)^2} = 12.60\ mm$$

Por lo tanto, el sistema siguiente quedaría completamente determinado en función de un parámetro de entrada: θ_1, θ_2 o β.

$$l_1 \cos\theta_1 + l_2 \cos(\theta_2 - \theta_1) = R \sin\beta + x_0$$

$$-l_1 \sin\theta_1 + l_2 \sin(\theta_2 - \theta_1) = R \cos\beta + y_0$$

Una vez determinadas todas las variables es posible construir la posición de la cámara del sensor Pixy en el sistema de referencia $(\boldsymbol{x}, \boldsymbol{y})$.

$$x_c = x_0 + l_6 \sin\left(\frac{\pi}{2} - \phi\right) + l_5 \cos\phi$$

$$y_c = y_0 + l_6 \cos\left(\frac{\pi}{2} - \phi\right) - l_5 \sin\phi$$

Además, la dirección del enfoque de la cámara puede expresarse a través del vector unitario siguiente:

$$\overrightarrow{v_c} = [\cos\phi\ , -\sin\phi\ ]$$

6.2 Modelado 3D del campo visual

El movimiento horizontal estará referenciado a un sistema de referencia $(\boldsymbol{x''}, \boldsymbol{y''}, \boldsymbol{z''})$ anclado al centro de giro del servomotor de movimiento horizontal. Una vez resuelto el problema vertical, sería posible determinar la posición tridimensional conociendo únicamente $(\boldsymbol{x_0}, \boldsymbol{y_0})$ en este sistema de referencia.

Es posible asociar la coordenada $\boldsymbol{z''}$ con $\boldsymbol{y'_c}$ ya que ambos ceros se encuentran en la misma cota. Así mismo, es posible trasladar a $(\boldsymbol{x''}, \boldsymbol{y''})$ mediante una traslación y un giro. Por lo tanto:

$$z''_c = l_6 \sin\left(\frac{\pi}{2} - \phi\right) + l_5 \cos\phi$$

$$x''_c = \left(x_0^* + l_6 \sin\left(\frac{\pi}{2} - \phi\right) + l_5 \cos\phi\right) \cos\psi$$

$$y''_c = \left(x_0^* + l_6 \sin\left(\frac{\pi}{2} - \phi\right) + l_5 \cos\phi\right) \sin\psi$$

Donde:

- $x_0^* = 21.47\ mm$

En la figura siguiente pueden observarse el sistema de referencia.

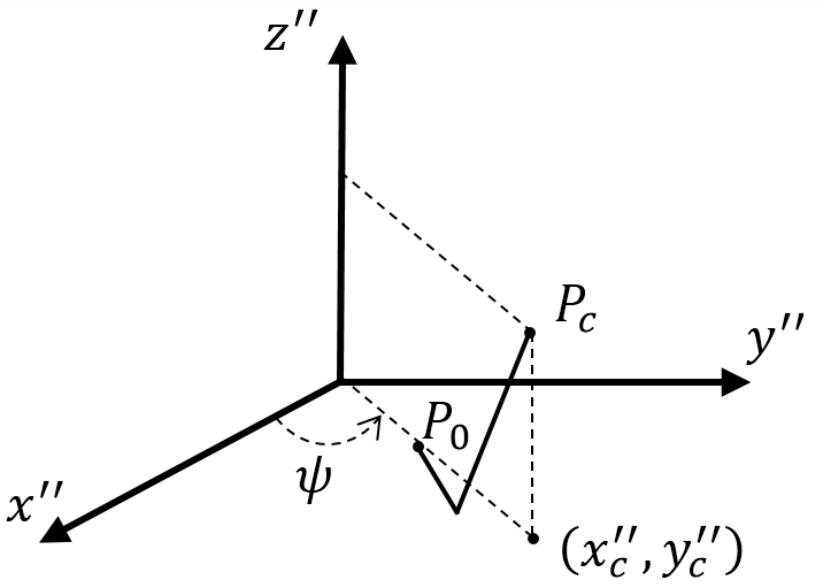


Figura 6-3. Sistema de referencia 3D

Previamente a la representación del campo visual de la cámara del sensor, es necesario establecer los rangos de las variables y la relación entre las unidades empleadas por Pixy para los ángulos y los grados sexagesimales.

Las variables de control del movimiento del sensor Pixy son los ángulos de los dos servomotores (θ_1, ψ), por lo tanto, según los sistemas de referencia anteriores sus intervalos de movimiento serían los siguientes:

$$\theta_1 \in [37.59\ °, 132.51\ °]$$

$$\psi \in [25\ °, 155\ °]$$

De donde es posible extraer que el rango para β:

$$\beta \in [-45\ °, 45\ °]$$

Una vez establecidos estos rangos es posible establecer unas relaciones sencillas que relacionen el rango empleado por Pixy y el de la notación usada en este modelo. Para ello es necesario tener en cuenta las equivalencias siguientes:

	Sexagesimal	**Pixy**		**Sexagesimal**	**Pixy**
$\boldsymbol{\theta_1}$	37.59	1000	ψ	25	0
$\boldsymbol{\theta_1}$	132.51	0	ψ	155	1000

Tabla 7. Equivalencias de ángulos

Por lo tanto, es posible establecer las relaciones siguientes:

$$\theta_1 = 132.51 - \frac{94.92}{1000}\theta_{Pixy}$$

$$\psi = 25 + \frac{130}{1000}\psi_{Pixy}$$

Finalmente, es posible representar el rango visual de la cámara del sensor Pixy haciendo uso de todas las relaciones anteriores.

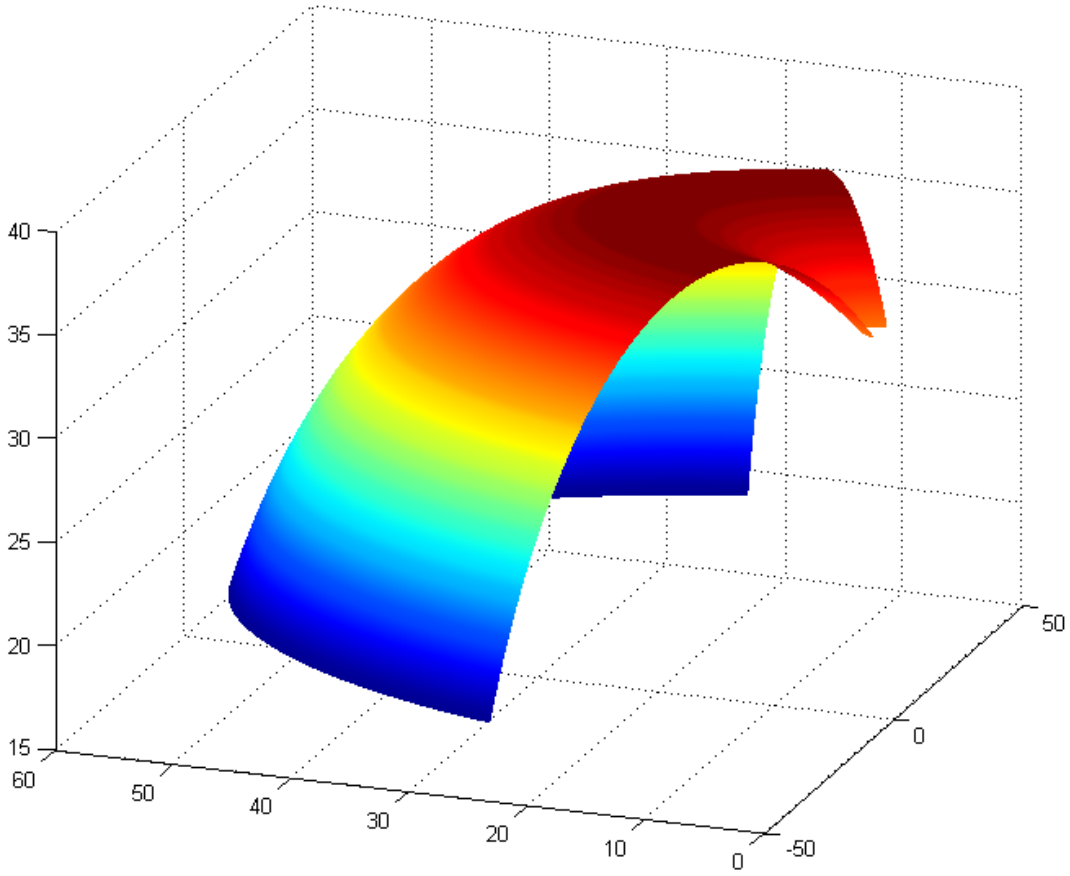


Figura 6-4. Campo visual del sensor Pixy

6.3 Algoritmo de detección: características y funcionamiento

En esta sección se presentará el funcionamiento del algoritmo de rastreo de elementos implementado en Arduino IDE. Este algoritmo hará uso del controlador PID para posicionar los elementos detectados en el centro del enfoque. Pero previamente será necesario presentar la trayectoria que recorrerá el sensor Pixy para barrer de forma efectiva todo su campo visual.

Para analizar el campo visual se ha tomado como parámetro la variable β y las relaciones que determinan la posición de la cámara del sensor en el sistema de referencia $(\boldsymbol{x}', \boldsymbol{y}')$.

$$x_c = l_6 \sin\left(\frac{\pi}{2} - \phi\right) + l_5 \cos\phi$$

$$y_c = l_6 \cos\left(\frac{\pi}{2} - \phi\right) - l_5 \sin\phi$$

En la figura siguiente es posible observar el recorrido de la cámara del sensor variando el parámetro β en todo el rango disponible.

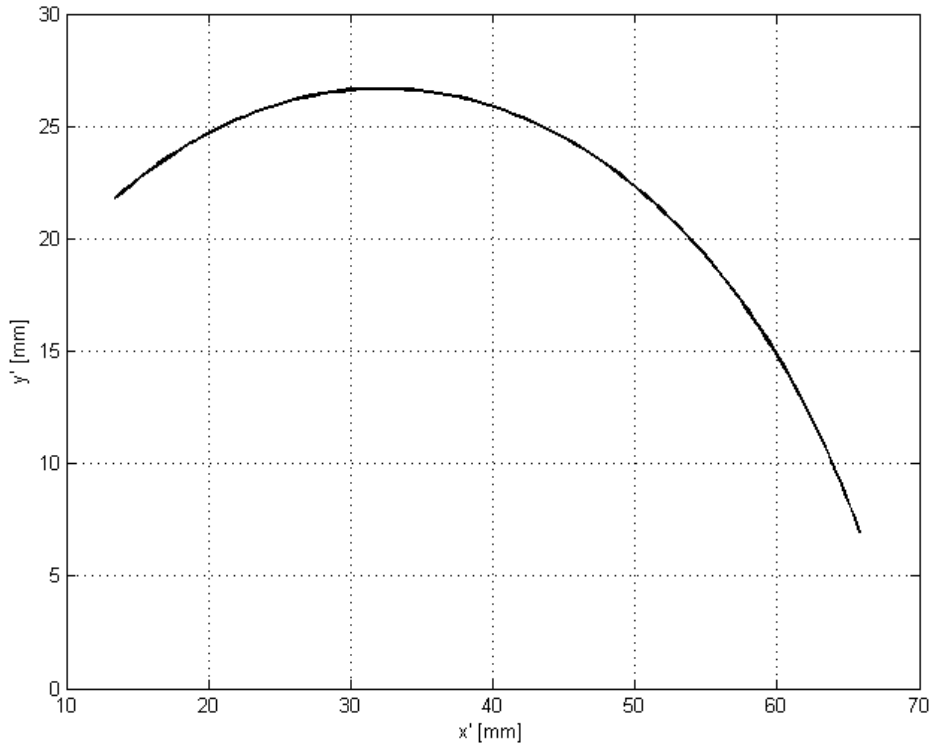


Figura 6-5. Recorrido de la cámara del sensor Pixy

El objetivo del barrido del campo visual es poder observar todo el campo visual desde posiciones discretas. De esta forma, se ha divido el recorrido en 4 puntos intermedios a modo de ejemplo. En la figura siguiente puede observarse a modo de vector rojo la dirección del enfoque, y con vectores azules la amplitud del enfoque.

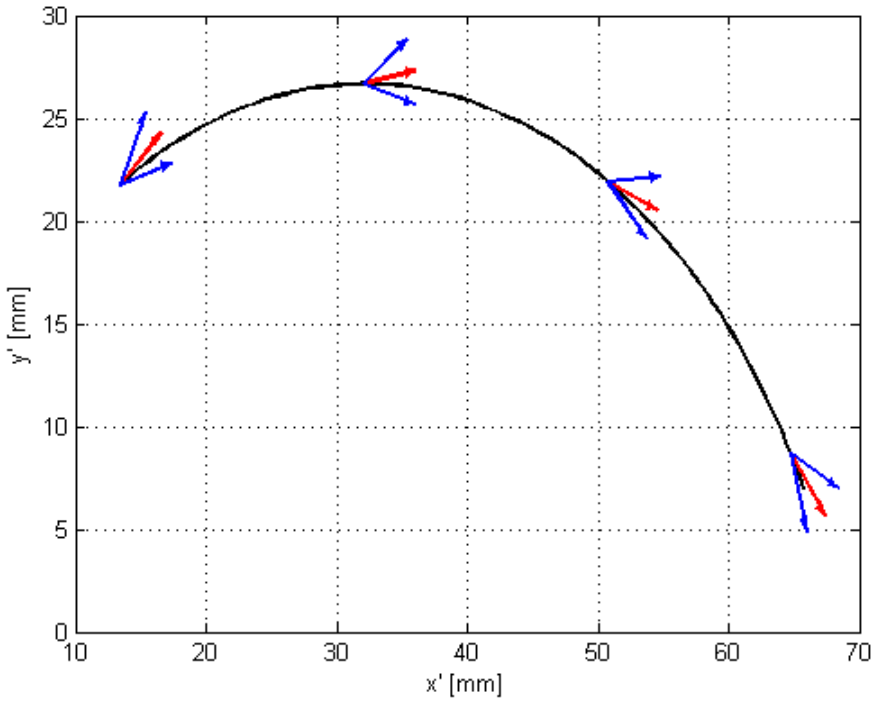


Figura 6-6. Recorrido junto con el enfoque

El código de arduino implementado permite realizar cualquier recorrido discreto deseable, pero se ha optado por dividir el recorrido en 5 recorridos verticales con 3 posiciones cada uno. En la tabla siguiente se

observan las posiciones de los recorridos.

Iteración	1	2	3
$\boldsymbol{\theta_{1,Pixy}}$	250	500	750

Tabla 8. Posiciones del recorrido vertical.

Iteración	1	2	3	4	5
$\boldsymbol{\psi_{Pixy}}$	0	250	500	750	1000

Tabla 9. Posiciones del recorrido horizontal

Se ha optado por comenzar el recorrido vertical en 250 pues no se prevé que existan elementos perceptibles desde esa posición para la práctica. Y se ha optado por finalizar en 750 pues una posición más reducida implicaría la existencia de elementos muy cerca. Estos parámetros pueden ser modificados de acuerdo a las condiciones que requiera el tipo de barrido y lo expuesto anteriormente es una generalidad que no tiene por qué ser el caso.

El código de Arduino empleado se encuentra en el Apéndice B. En adelante se presenta un resumen con aclaraciones sobre los bloques que presenta.

1. **Definición de variables y librerías**: en este primer bloque se definen las librerías y variables empleadas en el algoritmo.
2. **Funciones**: el algoritmo emplea 3 funciones distintas. Dos controladores PID y el bloque con la trayectoria de barrido empleada
3. **Inicialización**: se inicia el puerto serial y los valores de las variables iniciales.
4. **Bucle principal**: bucle con el contenido ejecutable.
    a. **Definición de variables**: se definen variables que serán de uso en el bloque del bucle principal.
    b. **Posición de rastreo**: modifica en cada iteración del bucle principal la posición de los servos de acuerdo a la trayectoria de barrido.
    c. **Inicialización del PID**: inicializa las variables empleadas en el PID en esta iteración.
    d. **Bucle de detección**: bucle que procesa la imagen de una posición de la trayectoria. Es necesario realizar iteraciones por problemas de refresco de la imagen del sensor y por retraso de comunicación.
    e. **Tipo de señales**: establece el tipo de señales de los elementos.
    f. **Bucle de posicionamiento de objetivos**: bucle que ejecuta el PID desde la posición del barrido para cada elemento y posiciona el objeto en el centro del enfoque.
    g. **Envío de información al puerto serial**: envía las posiciones de los servomotores de cada elemento junto con el error cometido.

7. IMPLEMENTACIÓN DE TRAYECTORIAS

En este capítulo se describirá la adaptación de las ecuaciones para giros desarrolladas en el Capítulo 2. Cinemática de la Actitud. Se discretizarán y se adaptarán a los parámetros de entrada. Así mismo, se presentara el algoritmo desarrollado en Arduino IDE que permitirá mover los servomotores del sensor Pixy de forma que ejecute las trayectorias.

El desarrollo del algoritmo de las trayectorias será generalista y estará enfocado de forma que se adapte a los requisitos del usuario. Permitirá concatenar tantas trayectorias como se desee, de cualquier tipo y con las características que se impongan. Estas trayectorias podrán realizarse entre las posiciones de los distintos elementos detectados por el algoritmo de detección.

Como se presentará en el próximo capítulo, la interacción con el usuario se desarrollará a través de unas interfaces de usuario creadas en MATLAB que evitarán que el usuario tenga que modificar directamente el código en Arduino IDE.

7.1 Adaptación de las ecuaciones a los parámetros de control

Arduino no permite modificar el par aplicado en los servomotores del sensor Pixy puesto que la conexión es entre la placa de Arduino y un puerto periférico. Esto plantearía un gran inconveniente ya que las ecuaciones que determinan los giros dependen del par aplicado. Para resolver este problema se ha hecho uso del parámetro (α), al expresar las ecuaciones en función de este parámetro se generaría una trayectoria a seguir que simularía las condiciones de distintos pares aplicados.

Así mismo, es necesario establecer un ángulo final θ_F e inicial θ_0 en las ecuaciones, de forma que la trayectoria vaya de un punto a otro deseado. La introducción de estos ángulos es igual para el movimiento vertical y el movimiento horizontal, pues se han considerado desacoplados.

Finalmente se ha optado por emplear dos parámetros de entrada: la duración de la trayectoria (T) y el parámetro (α). Previo a la presentación de las ecuaciones adaptadas a Arduino IDE se necesario establecer la relación entre los pares aplicados y los parámetros de control.

De la expresión del parámetro α es posible obtener la relación entre el par máximo aplicado y el par mínimo aplicado:

$$\alpha = \frac{-M_{decel}}{M_{acel} - M_{decel}}$$

$$M_{decel} = \frac{\alpha}{\alpha - 1} M_{acel}$$

$$M_{acel} = \frac{\alpha - 1}{\alpha} M_{decel}$$

Si se combinan las expresiones anteriores con la expresión de la duración de la trayectoria con pares desiguales se obtendrían las relaciones siguientes:

$$T^2 = 2I\theta_F \frac{M_{acel} - M_{decel}}{-M_{acel} M_{decel}}$$

$$\frac{M_{acel}}{I} = \frac{2\theta_F}{\alpha T^2}$$

$$\frac{M_{decel}}{I} = \frac{2\theta_F}{(\alpha - 1)T^2}$$

7.1.1 Maniobra de tiempo mínimo

La maniobra de tiempo mínimo quedaría expresada en función de los parámetros de entrada de la forma siguiente:

$$\theta(t) = \begin{cases} \theta_0 + \frac{\theta_F - \theta_0}{\alpha T^2} t^2 \ , & 0 \leq t \leq \alpha T \\ \theta_0 + (\theta_F - \theta_0)\left[\alpha + \frac{2(t-\alpha T)}{T} + \frac{(t-\alpha T)^2}{(\alpha-1)T^2}\right], & \alpha T \leq t \leq T \end{cases}$$

7.1.2 Maniobra de mínima energía

La maniobra de mínima energía quedaría expresada en función de los parámetros de entrada de la forma siguiente:

$$\theta(t) = \theta_0 + \frac{\theta_F - \theta_0}{T^3} t^2 (3T - 2t)$$

7.2 Algoritmo de trayectorias: características y funcionamiento

En esta sección se presentará el funcionamiento del algoritmo que permite al sensor Pixy recorrer las trayectorias deseadas. Este código será generado por la interfaz de usuario implementada en MATLAB e incluirá las características impuestas por el usuario desde la interfaz.

El código de Arduino empleado se encuentra en el Apéndice B. En adelante se presenta un resumen con aclaraciones sobre los bloques que presenta.

1. **Definición de variables y librerías**: en este primer bloque se definen las librerías y variables empleadas en el algoritmo.
2. **Inicialización**: se inicia el puerto serial y los valores de las variables iniciales.
3. **Bucle principal**: bucle con el contenido ejecutable.
    a. **Definición de variables**: se definen variables que serán de uso en el bloque del bucle principal.
    b. **Trayectoria de tiempo mínimo**: calcula la posición de los servomotores en caso de que la trayectoria a ejecutar sea de tiempo mínimo.
    c. **Trayectoria de mínima energía**: calcula la posición de los servomotores en caso de que la trayectoria a ejecutar sea de mínima energía.
    d. **Posicionado de los servomotores**: ejecuta las órdenes de posicionado de los servomotores en función de la trayectoria seleccionada.

Es necesario hacer constar la complejidad para medir la máxima velocidad con la que los servomotores pueden desplazarse con alta precisión. Por lo tanto, se ha establecido un límite inferior para la duración de las trayectorias de 1 segundo.

$$T \leq 1\ s$$

8. PROCESAMIENTO DE DATOS E INTERFACES

En este capítulo se describirá la metodología desarrollada para transferir la información de los elementos detectados desde la placa de Arduino a un ordenador. Una vez expuesta la metodología se procederá a presentar las interfaces de usuario. Estas interfaces pretenden facilitar el procesamiento de la información obtenida representando la información y generando un archivo para Arduino IDE que ejecute las trayectorias deseadas.

Además, se ha desarrollado una interfaz que permite tratar imágenes capturadas por el sensor Pixy para detectar elementos. El objetivo de esta interfaz es ilustrar de forma sencilla el procesamiento de imágenes de satélites en su objetivo de detectar elementos.

8.1 Programa de captura de datos y formato del archivo

Para capturar la información enviada por la placa de Arduino se hará uso de la aplicación de escritorio CoolTerm. Esta aplicación puede encontrarse en internet aunque carece de sitio oficial. Esta aplicación es capaz de conectarse al puerto serial y grabar la información obtenida en un archivo .txt para su posterior procesado.

El proceso para realizar la captura de información es el siguiente:

1. Cargar el programa de detección de elementos en el sensor Pixy.
2. Asegurar que la aplicación de escritorio de Arduino IDE no se encuentra conectada al puerto serial.
3. Ejecutar la aplicación CoolTerm y conectar al puerto serial.
4. Iniciar la captura de datos en un archivo .txt.
5. Iniciar el algoritmo de detección de elementos desde PixyMon.
6. Una vez finalizada la detección desconectar el programa.

Para conectar la aplicación CoolTerm al puerto serial es necesario pulsar la opción conectar de la barra de herramientas de la aplicación.

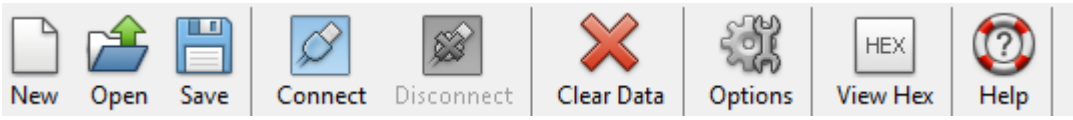


Figura 8-1. Barra de herramientas de CoolTerm

Para confirmar que se ha establecido la conexión con el puerto serial se deben iluminar los botones de la barra inferior de la aplicación de forma similar a la que se puede observar en la siguiente figura.

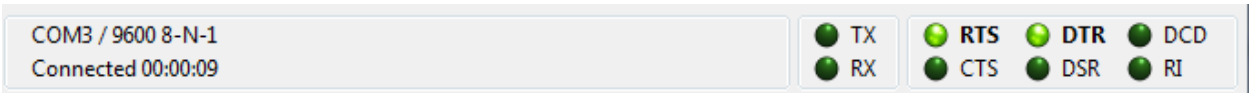


Figura 8-2. Confirmación de conexión

Para grabar la información en un archivo .txt es necesario crear primero ese mismo archivo desde la aplicación, tal y como se puede observar en la figura siguiente.

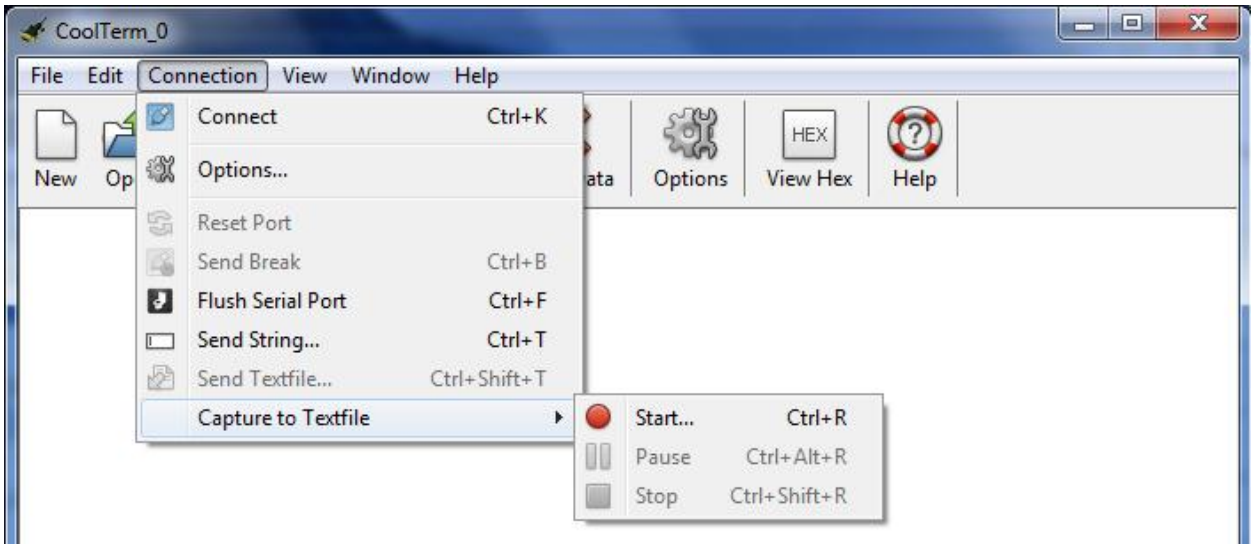


Figura 8-3. Captura de datos en un archivo .txt

Los datos se almacenarán en un archivo de texto con la siguiente organización. Cada detección realizada se ubicará en una fila distinta y consecutiva a la anterior. La información de cada detección será la siguiente: número de la señal, posición del servomotor horizontal, posición del servomotor vertical, error horizontal y error vertical. La información estará separada por comas y la línea acabará con un punto y coma. Cabe destacar que el puerto serial puede enviar información errónea que se traduce en una línea con el texto *cs error* escrito.

En la figura siguiente puede observarse un ejemplo de información obtenida con esta metodología.

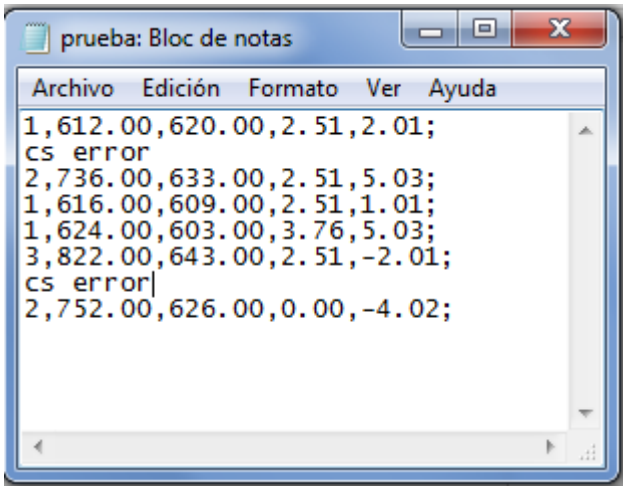


Figura 8-4. Ejemplo de información obtenida con CoolTerm

8.2 Interfaces de usuario

Las interfaces de usuario han sido creadas en MATLAB empleando GUI. GUI es una funcionalidad de MATLAB que permite la creación de apps con interfaces gráficas de usuario. Estas aplicaciones permiten una interacción sencilla sin necesidad de trabajar con los scripts y las funciones de MATLAB.

8.2.1 Interfaz de trayectorias

El principal objetivo de esta interfaz de usuario es procesar el archivo .txt con la información de los elementos detectados para así representar la información de forma que permita al usuario establecer las trayectorias deseadas.

Esta interfaz cuenta con tres bloques diferenciados que se explican a continuación:

1. **Lectura y visualización**: este bloque permite leer y cargar la información obtenida del algoritmo de detección de elementos. El pulsador *Leer datos* permite cargar la información y representarlas en el gráfico.
2. **Generador de trayectorias**: Permite introducir el orden del recorrido a realizar con las características en tiempo y par deseadas. Además genera un archivo con el código necesario para ejecutar en Arduino IDE.
3. **Gráfico**: Representa el campo visual del sensor Pixy y la ubicación de los elementos en el mismo.

En la figura siguiente puede observarse el entorno de la interfaz gráfica inicial.

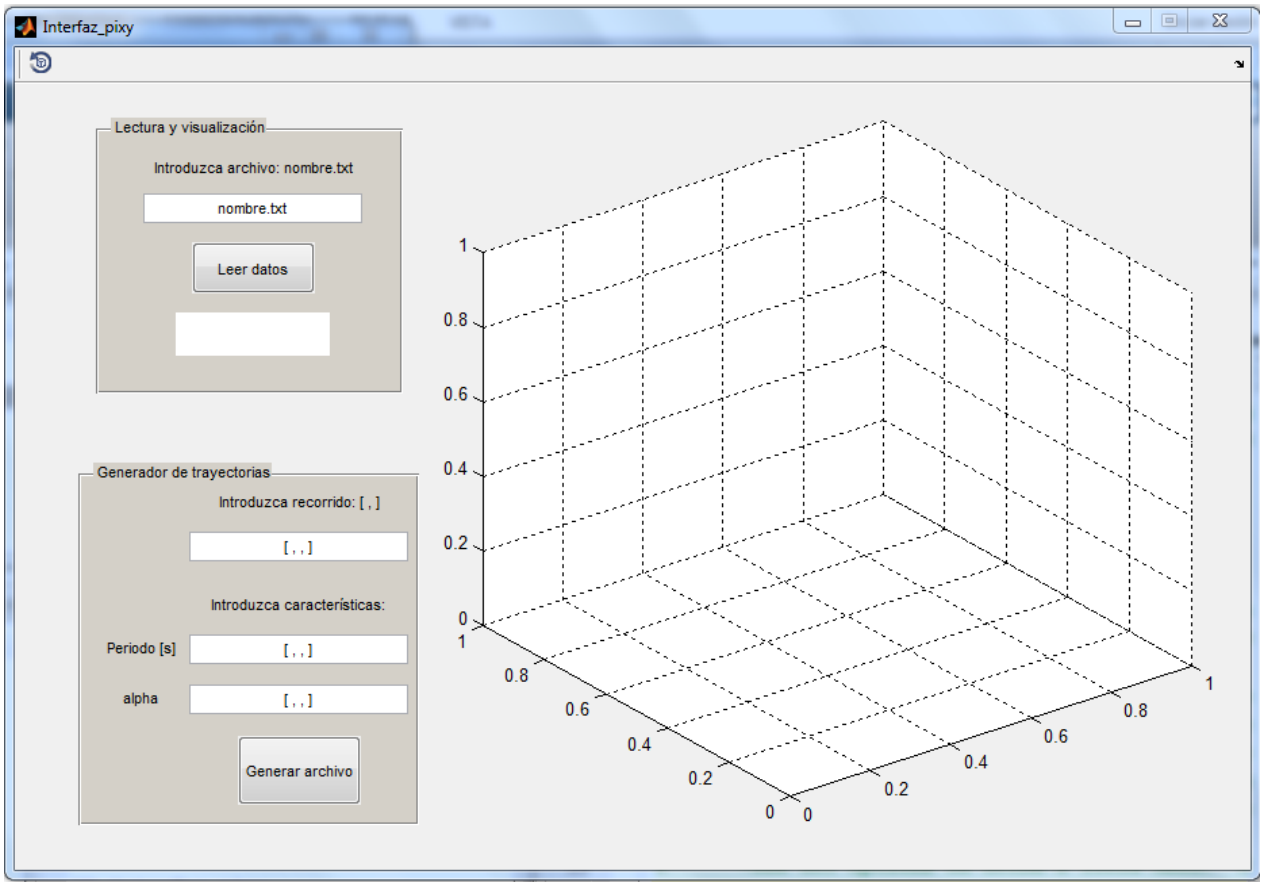


Figura 8-5. Interfaz de usuario para trayectorias: inicial

En la figura siguiente puede observarse el entorno gráfico con la información cargada y representada.

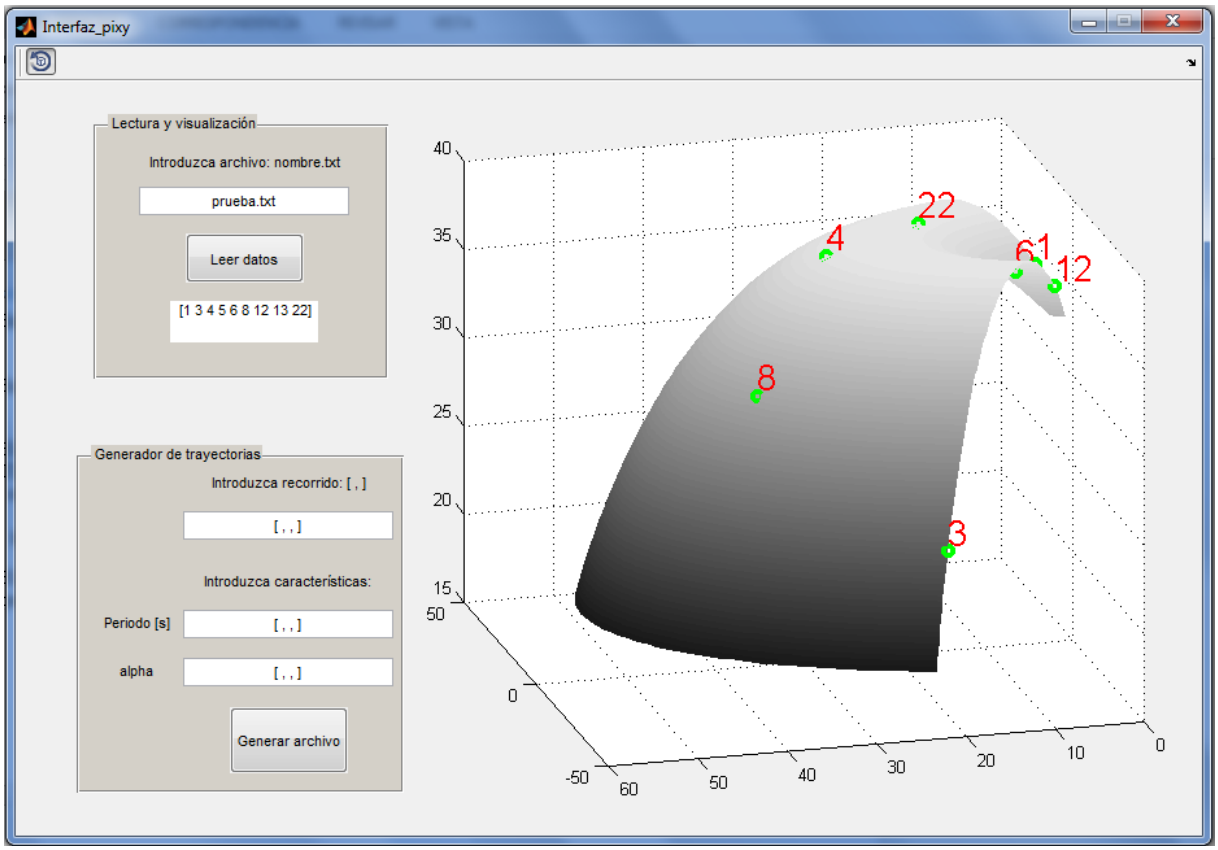


Figura 8-6. Interfaz de usuario para trayectorias: carga de datos

En la figura siguiente puede observarse un ejemplo de introducción de trayectorias.

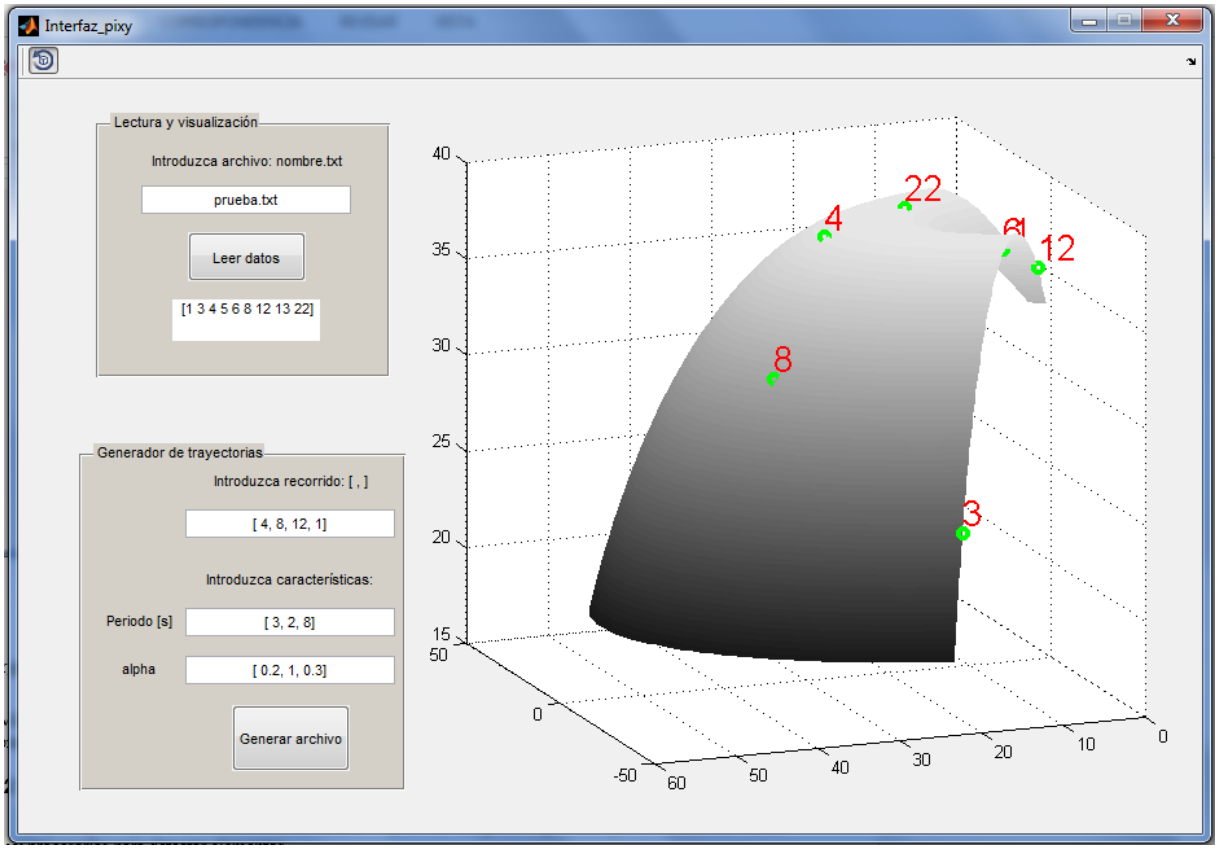


Figura 8-7. Interfaz de usuario para trayectorias: ejemplo trayectorias

La trayectoria por defecto es la de tiempo mínimo, por lo que para establecer una trayectoria de mínima energía se ha definido para un valor del parámetro $\alpha = 1$.

El código en MATLAB que genera la interfaz se encuentra en el Apéndice C. Cabe destacar que el código en MATLAB no permite generar la interfaz, para ello sería necesario crear una figura con el comando *guide* tal y como se observa en las figuras anteriores.

8.2.2 Interfaz de visión

El principal objetivo de esta interfaz de usuario es procesar imágenes capturadas por la cámara del sensor Pixy y procesarlas para detectar elementos.

Esta interfaz cuenta con dos bloques diferenciados que se explican a continuación:

1. **Carga de imagen**: este bloque permite la carga y representación de la imagen a procesar.
2. **Procesado**: este bloque permite el procesado de la imagen con los siguientes pasos: binarizado de la imagen, modificación del umbral, eliminación del ruido, rellenado de huecos, corrección de los bordes según la forma y posicionado el centroide el elemento.
3. **Gráfico**: este bloque muestra la imagen tratada.

En la figura siguiente se puede observar la interfaz de visión inicial.

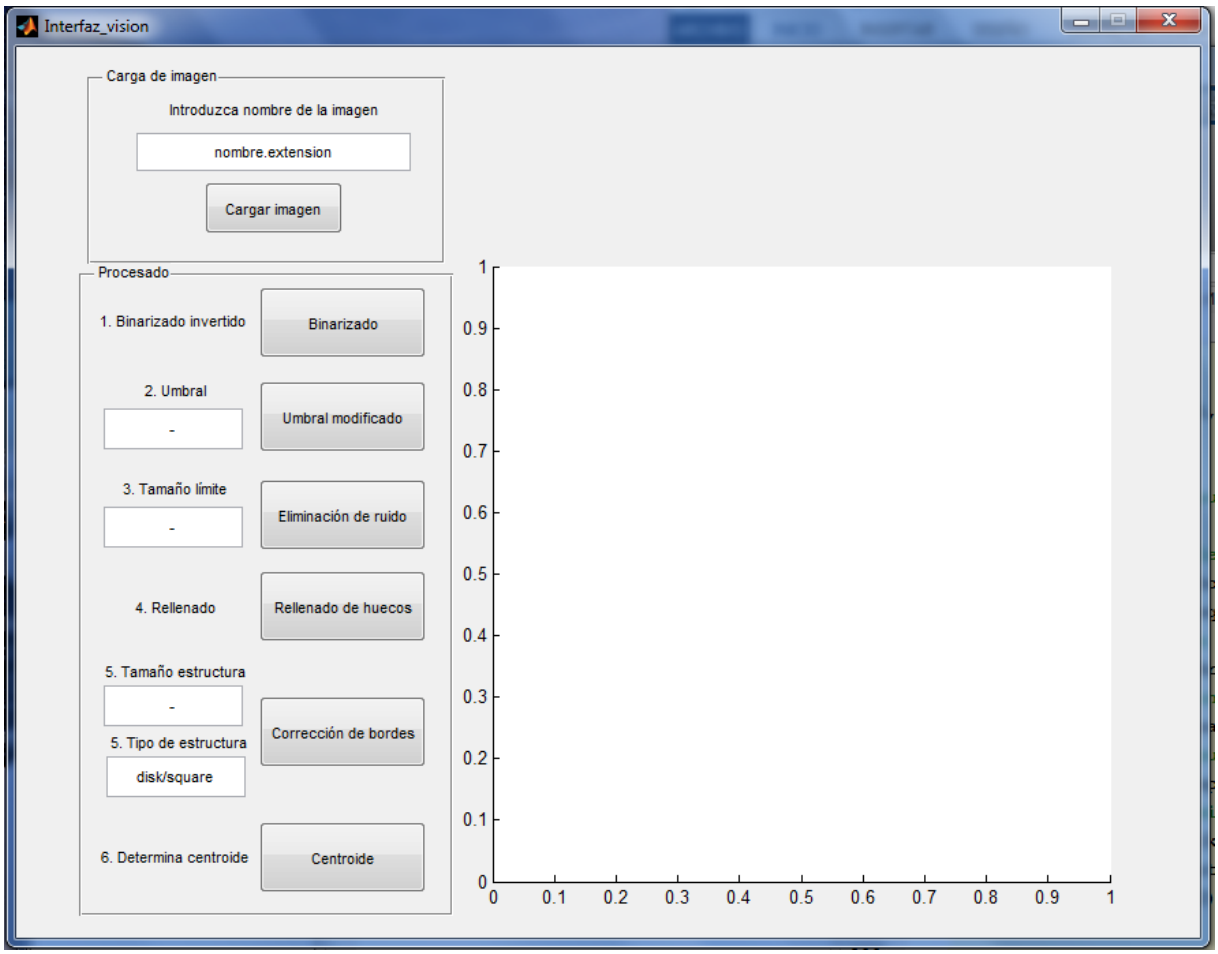


Figura 8-8. Interfaz de usuario para visión: inicial

En la figura siguiente se puede observar un ejemplo de imagen con binarizado invertido. Dado que las funciones posteriores trabajan con el color blanco, se ha optado por representar como 1 – binarizado.

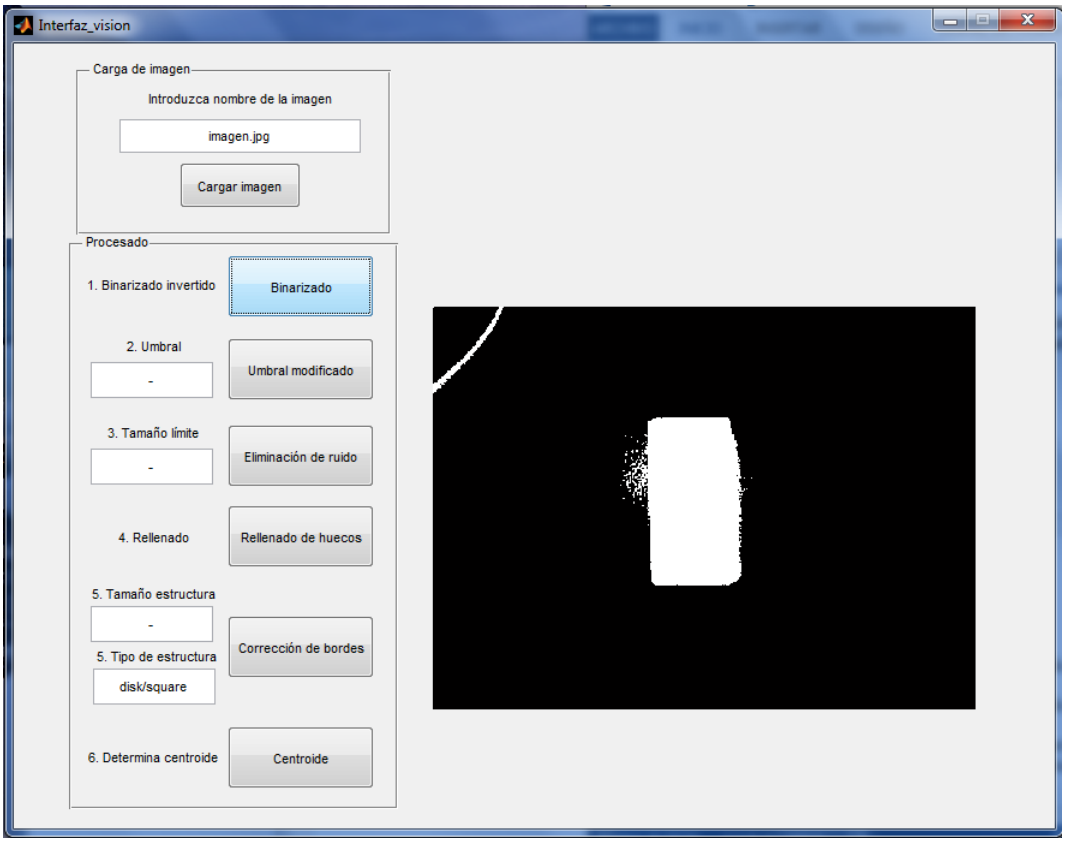


Figura 8-9. Interfaz de usuario para visión: binarizado

En la figura siguiente se puede observar el resultado de aplicar los pasos 2 a 5.

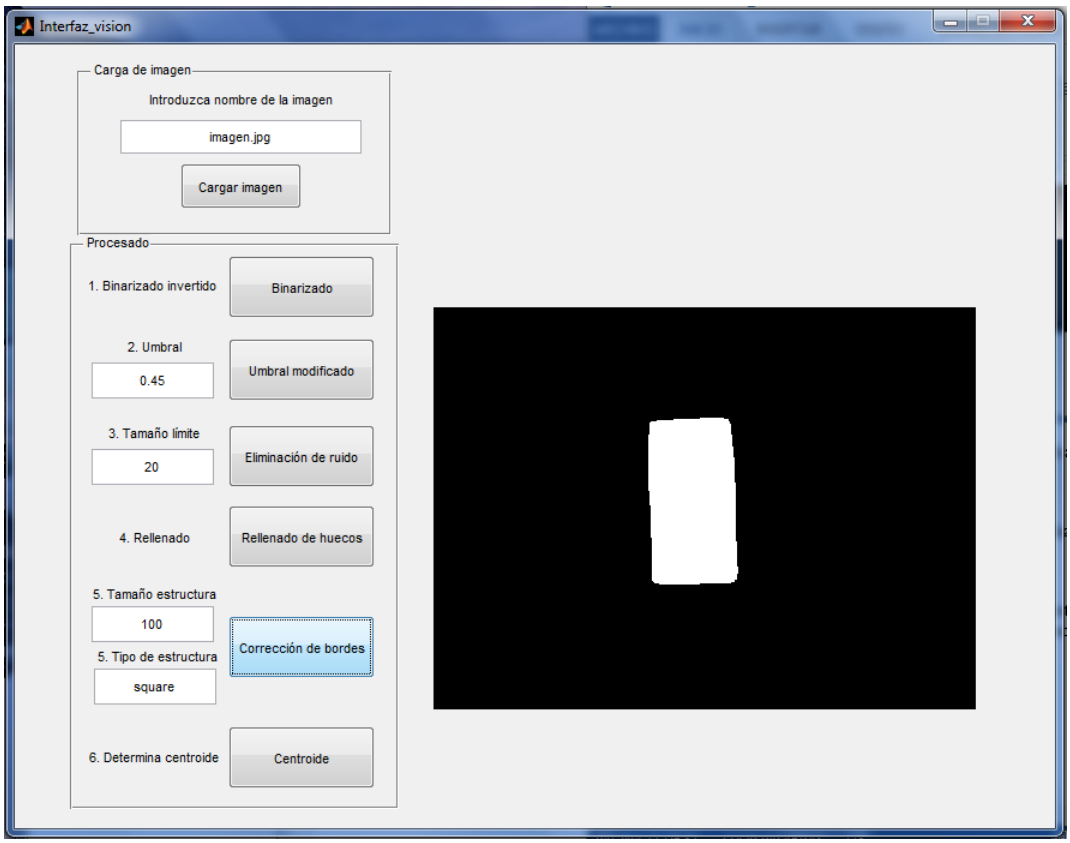


Figura 8-10. Interfaz de usuario para visión: pasos 2 a 5

En la figura siguiente se puede observar el resultado final del procesado.

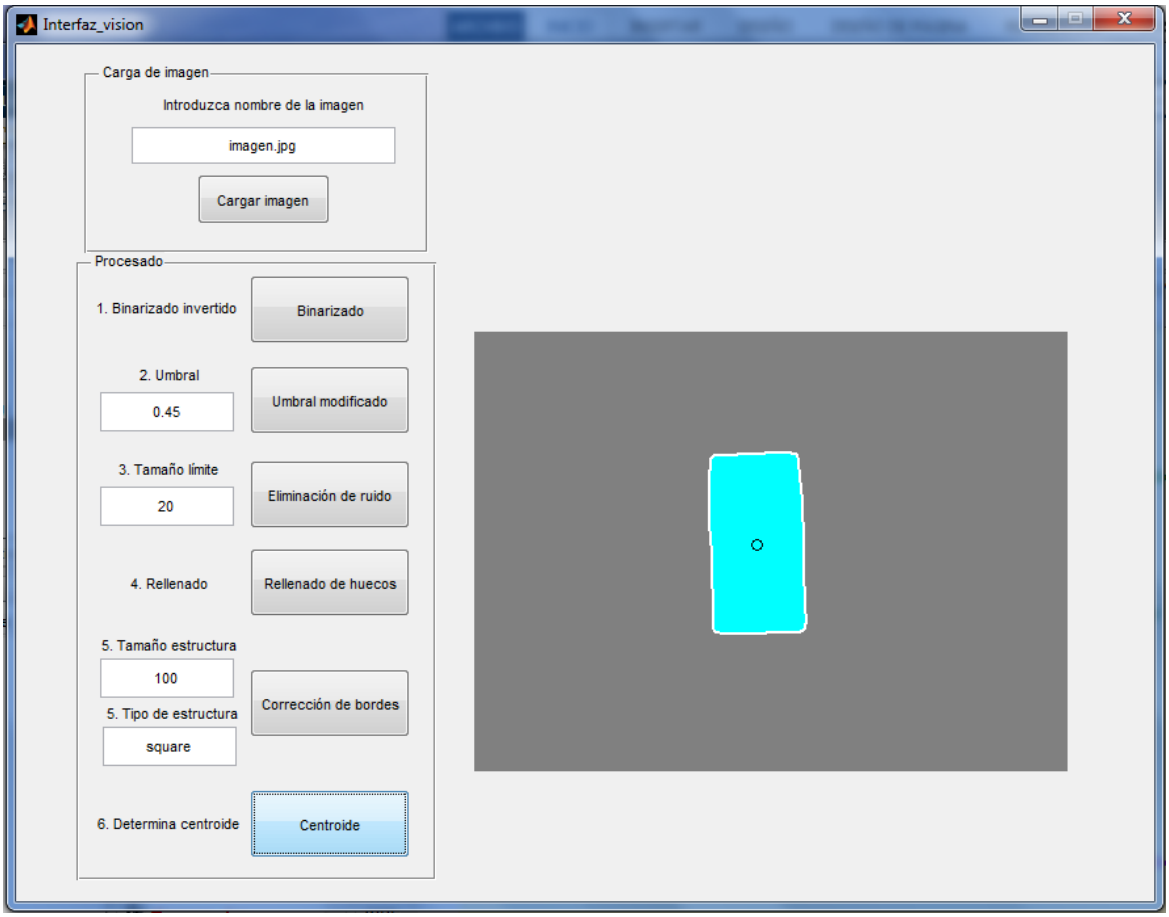


Figura 8-11. Interfaz de usuario para visión: centroide

El umbral es un valor entre 0 y 1 que asigna el color blanco o negro tomando como punto de corte este umbral. Los valores tamaño límite y tamaño estructura son tamaños de píxeles. En el primer caso el tamaño límite a partir del cual se eliminan elementos y el segundo caso el tamaño de la estructura de referencia para suavizar los bordes.

El código en MATLAB que genera la interfaz se encuentra en el Apéndice C. Cabe destacar que el código en MATLAB no permite generar la interfaz, para ello sería necesario crear una figura con el comando *guide* tal y como se observa en las figuras anteriores.

9. PRÁCTICA

El principal objetivo de este proyecto es la creación de material para el desarrollo de una práctica para los alumnos de la asignatura *Dinámica de Vehículos Espaciales* del Máster en Ingeniería Aeronáutica de la Universidad de Sevilla. Para ello, se ha desarrollado un boletín de prácticas que se encuentra en el Anexo A de este documento. Este boletín pretende ser lo más didáctico y sencillo posible para el alumno, pues el objeto principal de esta práctica es la familiarización con la orientación de objetos y los cambios de actitud, no la programación en sí.

En este capítulo se presentará el montaje final realizado para la práctica y la esquematización del boletín de prácticas con las tareas y procesos que el alumno desarrollará en el progreso de la práctica. Dado el carácter de la práctica, el boletín contará exclusivamente con una guía sin necesidad de entregar resultados.

9.1 Montaje

Los elementos con los que el alumno contará para el desempeño de la práctica son los siguientes:

- **Conjunto sensor Pixy y Arduino**: montaje que une ambos elementos para facilitar su uso.
- **Panel doble orientable**: a modo de pantalla para evitar interferencias con los elementos a detectar.
- **Cable USB tipo mini B**: cable que permite comunicarse con el sensor Pixy.
- **Cable USB tipo B**: cable que permite comunicarse con la placa de Arduino UNO.
- **Conjunto de elementos detectables**: serie de objetos de colores claros y distintos que facilitará el reconocimiento.

El conjunto sensor Pixy y Arduino consiste en un montaje donde tanto el sensor Pixy como la placa de Arduino UNO se encuentran atornillados a una tabla de madera de $23x14.5x1$ cm. El principal objetivo de este montaje es evitar los movimientos indeseados de la base del sensor Pixy por motivos de inercia cuando se mueven los servomotores rápidamente. En las figuras siguientes puede observarse el conjunto.

Figura 9-1. Montaje del conjunto 1

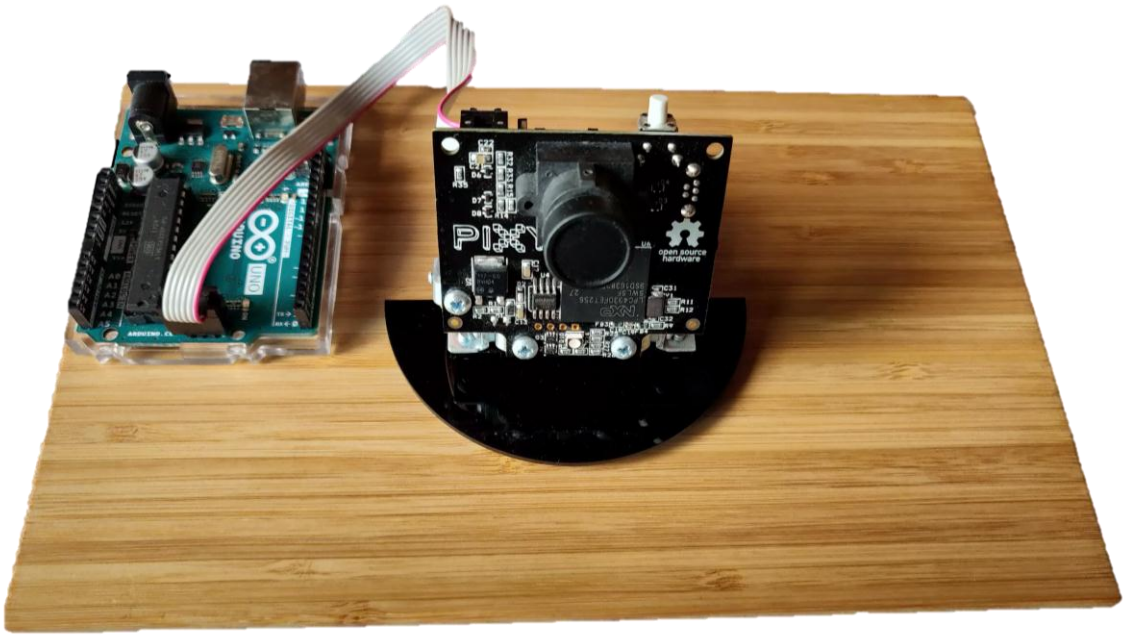


Figura 9-2. Montaje del conjunto 2

El panel doble orientable consiste en dos paneles de madera unidos con una bisagra que sirve a modo de fondo para el sensor. En la figura siguiente puede observarse este panel.

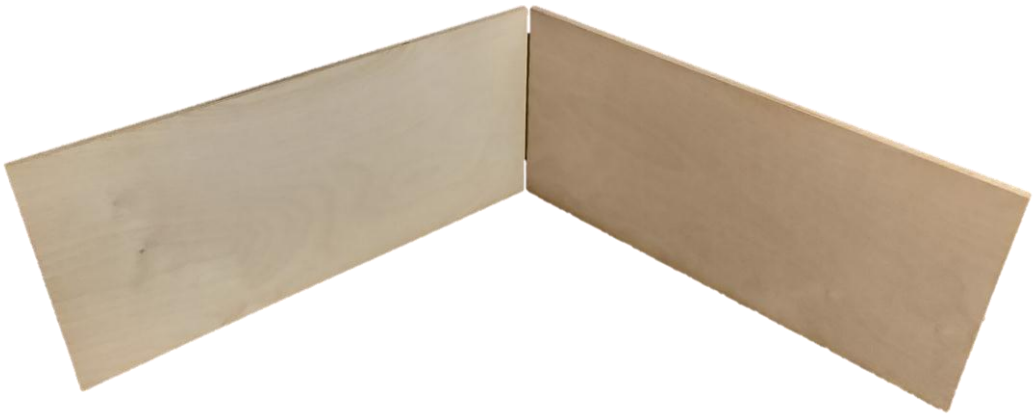


Figura 9-3. Panel doble de fondo

9.2 Boletín de la práctica

Inicialmente se presenta una breve introducción a la práctica con los objetivos y las tareas a realizar por el alumno. Posteriormente se presentarán las nociones básicas de la cinemática de la actitud, de Arduino, del sensor Pixy y de la aplicación PixyMon. Seguidamente se proporciona una explicación de todos los archivos de Arduino y MATLAB proporcionados al alumno, así como de la aplicación CoolTerm que permite la conexión a través del puerto serial con el ordenador. Finalmente se detallarían los pasos a seguir durante la práctica: se comenzaría con una familiarización con PixyMon y el aprendizaje de señales, se continuaría con el posicionado de elementos a detectar y la ejecución del programa de detección de elementos, se procedería al procesado de la información y se finalizaría con el uso de la interfaz de usuario para generar las trayectorias. Así mismo se añade la parte de tratado de visión.

El boletín que se ha descrito se puede encontrar en el Apéndice A de este documento.

10. CONCLUSIONES

El objeto del presente proyecto es la generación del material necesario para el desarrollo de una práctica que ilustre la cinemática de la actitud haciendo uso de un sensor visual con dos grados de libertad. Haciendo uso de un microcontrolador y de MATLAB ha sido posible la creación de una práctica que no exige al alumno conocimientos muy específicos para su desempeño y aprendizaje.

Una vez desarrollados los algoritmos, interfaces y material necesarios para la puesta a punto de la práctica, se finaliza este proyecto con una presentación de las conclusiones obtenidas del proyecto así como una lista con posibles mejoras y líneas de trabajo futuras.

10.1 Conclusiones

En esta sección se presentan las conclusiones obtenidas con respecto al desarrollo del proyecto y a los distintos elementos empleados para el desarrollo de la práctica. Las conclusiones son las siguientes:

- Se puede considerar comprobado que el uso del sensor Pixy ha proporcionado grandes ventajas para la consecución de los objetivos de este proyecto. La facilidad con la que el sensor puede ser controlado ha resultado ser de gran utilidad para la implementación de giros y trayectorias. Además, su capacidad para detectar elementos de distintos colores permite tratar directamente con distintos objetos sin necesidad de desarrollar un algoritmo que los diferencie.
- El uso de Arduino puede considerarse otra ventaja del uso del sensor Pixy, pues una plataforma de hardware y software libre muy versátil. Su lenguaje de programación sencillo ha permitido la implementación de un controlador PID haciendo uso de estamentos simples y sin necesidad de crear librerías propias. Además, el microcontrolador es capaz de transmitir la información del sensor Pixy directamente al puerto serial de un ordenador.
- El uso del controlador PID ha resultado ser bastante efectivo en cuanto que permite posicionar rápidamente los elementos detectados en el centro del enfoque sin dificultad. Así mismo, a pesar de haber sido ajustado de forma manual, no presenta oscilaciones significativas.
- El uso de MATLAB para el procesamiento de la información obtenida del algoritmo de rastreo y la creación de interfaces de usuario con GUI ha supuesto una mejora sustancial en la capacidad didáctica de la práctica. El uso de MATLAB es conocido por los alumnos de la asignatura y las interfaces GUI son intuitivas y de uso muy simple, con lo que la práctica se centra más en el propio contenido a aprender que en los pasos a seguir.
- Una de las mayores problemáticas de la práctica es la detección del sensor Pixy ya que puede dar falsos positivos y aún con una buena iluminación se producen fluctuaciones en la detección de los elementos. Así mismo, al apagar al desconectar el sensor Pixy de la alimentación, almacena información residual del anterior programa por defecto y puede llegar a tener un comportamiento errático inicialmente.
- Para hacer completamente viable la práctica para todos los alumnos de la asignatura *Dinámica de Vehículos Espaciales* sería necesario la adquisición de una mayor cantidad de sensores Pixy y de placas de Arduino, así como el resto del material necesario. Lo cual ocasiona que la mejora pedagógica de la práctica sea dependiente del presupuesto.

10.2 Mejoras y trabajos futuros

Aunque los objetivos del proyecto se han satisfecho completamente cabe tanto el planteamiento de posibles mejoras en el desarrollo de este proyecto como posibles ampliaciones futuras. Las posibles mejoras que podrían implementarse son las siguientes:

- Sería posible mejorar las hipótesis de la dinámica de la actitud que proporciona las ecuaciones de los giros. Con un mejor modelado de la inercia del conjunto, sería posible establecer unas trayectorias que aproximen mejor el tiempo mínimo. Como primera aproximación sería posible modelar en 3D el conjunto y asignar distintas masas para así conocer la inercia en cualquier sistema de referencia.
- Sería posible realizar un ajuste más preciso del controlador PID, empleando métodos empíricos de ajuste que permitan un mejor control del comportamiento de los servomotores.
- Sería posible mejorar las trayectorias seguidas si se conectasen los servomotores directamente a una placa de Arduino, para así poder aplicar señales de pulso modulado que regulen la velocidad angular.

Los trabajos futuros a realizar son los siguientes:

- Una posible línea de investigación futura consistiría en la implementación de un método TRIAD haciendo uso del sensor Pixy. Con un modelado fiel del sistema de referencia del sensor de la cámara e información externa de la posición de los elementos sería posible conocer la actitud del conjunto respecto a una referencia.
- Una posible línea de investigación futura consistiría en realizar trayectorias óptimas con elementos en movimiento o con el sensor Pixy unido a una base móvil que se desplace y rote para perseguir objetivos.

APÉNDICE A. BOLETÍN DE PRÁCTICAS

DINÁMICA DE VEHÍCULOS ESPACIALES

Práctica: Detección de elementos y apuntado. Maniobras de tiempo mínimo y de mínima energía. Procesado de imágenes.

En esta práctica se empleará una cámara orientable con capacidad para detectar objetos con el objetivo de ilustrar los conceptos de representación de la actitud y de la cinemática de la actitud aprendidos en la asignatura. Para ello se emplearán Arduino, MATLAB y PixyMon.

1. Cinemática de la actitud

Para el desarrollo de la práctica será necesario recordar ciertos aspectos de la cinemática de la actitud cursados en la asignatura. Comenzamos por el ángulo y el eje de Euler. El Teorema de Euler establece que: "el movimiento más general posible de un sólido con un punto fijo es una rotación alrededor de un único eje".

Por lo tanto, es posible representar la actitud de un sistema de referencia S respecto a otro sistema de referencia S' haciendo uso únicamente de un eje y un ángulo. Este eje y este ángulo serían:

- Eje de Euler: $\vec{e}_{S/S'}$
- Ángulo de Euler: θ

El eje de Euler se trata de un vector unitario, por lo que se puede escribir: $\vec{e}^{S}_{S/S} = [e_x \quad e_y \quad e_z]^T$, donde debe cumplirse: $e_x^2 + e_y^2 + e_z^2 = 1$.

Esto se traduce en que el eje de Euler es la dirección entorno a la que se debe girar un cierto ángulo para hacer coincidir el sistema de referencia S con el sistema de referencia S'. De esta forma, la actitud queda representada con los parámetros $(\vec{e}_{S/S'}, \theta)$.

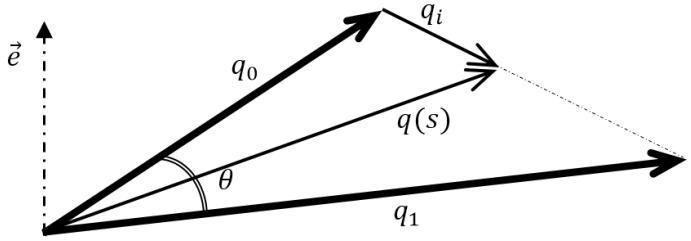
Dada la actitud de un sistema de referencia a través del eje de Euler y del ángulo de Euler $(\vec{e}, \theta)$, es posible codificar dicha actitud en forma de cuaterniones mediante las relaciones siguientes:

$$q = \begin{bmatrix} \cos(\theta/2) \\ \vec{e}\,\sin(\theta/2) \end{bmatrix}, \qquad |q| = 1$$

Si recordamos el cuaternión de interpolación, las actitudes intermedias entre dos dadas puede expresarse como:

$$q(s) = q_0 \star q_i = q_0 \star \begin{bmatrix} \cos(s\theta/2) \\ \vec{e}\,\sin(s\theta/2) \end{bmatrix}, \qquad donde\ s\epsilon[0,1]$$

En la figura siguiente quedaría ilustrado de forma esquemática y simplificada el proceso anterior:



Haciendo uso del concepto de cuaternión de interpolación podemos obtener el camino más corto entre A y B. Simplemente habría que determinar el cuaternión de interpolación, y de dicho cuaternión sería posible obtener el eje entorno al cual sería necesario realizar la rotación y el ángulo que sería necesario girar.

$$q_R = q_0^* \star q_1 = \begin{bmatrix} \cos(\theta/2) \\ \vec{e}\sin(\theta/2) \end{bmatrix}$$

Una vez determinada la dirección de giro, sería posible establecer la velocidad angular en cada instante de tiempo:

$$\vec{\omega} = \omega(t)\vec{e}$$

Por lo tanto, dadas una actitud inicial y una final, el problema se vería reducido a determinar la variación de la velocidad angular de forma que se satisfagan las condiciones impuestas. Para ello, se recurrirá a una dinámica simplificada bajo la hipótesis de que el eje de giro es uno de simetría de masa. Por lo tanto:

$$\dot{\theta} = \omega$$

$$I\omega \doteq M$$

Tal y como se ha visto en la asignatura podríamos considerar dos maniobras distintas: maniobra de tiempo mínimo y maniobra de mínima energía.

1.1. Maniobra de pares asimétricos

La evolución del ángulo para esta maniobra puede expresarse como sigue:

$$\theta(t) = \begin{cases} \dfrac{M_{acel}}{I}\dfrac{t^2}{2}\ , & 0 \le t \le \alpha T \\ \dfrac{M_{acel}}{I}\dfrac{(\alpha T)^2}{2} + \dfrac{M_{acel}}{I}\alpha T(t-\alpha T) + \dfrac{M_{decel}}{I}\dfrac{(t-\alpha T)^2}{2}, & \alpha T \le t \le T \end{cases}$$

Será necesario generalizar la relación anterior para considerar un ángulo inicial distinto de 0 y haciendo uso de las relaciones siguientes es posible expresar la expresión anterior en función únicamente de α y de T.

$$\alpha = \frac{-M_{decel}}{M_{acel} - M_{decel}}$$

$$T^2 = 2I\theta_F \frac{M_{acel} - M_{decel}}{-M_{acel}M_{decel}}$$

$$\frac{M_{acel}}{I} = \frac{2\theta_F}{\alpha T^2}$$

$$\frac{M_{decel}}{I} = \frac{2\theta_F}{(\alpha - 1)T^2}$$

Por lo que se obtendría:

$$\theta(t) = \begin{cases} \theta_0 + \dfrac{\theta_F - \theta_0}{\alpha T^2}t^2\ , & 0 \le t \le \alpha T \\ \theta_0 + (\theta_F - \theta_0)\left[\alpha + \dfrac{2(t-\alpha T)}{T} + \dfrac{(t-\alpha T)^2}{(\alpha-1)T^2}\right], & \alpha T \le t \le T \end{cases}$$

1.2. Maniobra de mínima energía

La evolución del ángulo para esta maniobra puede expresarse como sigue:

$$\theta(t) = \frac{\theta_F}{T^3}t^2(3T - 2t)$$

Será necesario generalizar la relación anterior para considerar un ángulo inicial distinto de 0, por lo tanto la expresión anterior resultaría:

$$\theta(t) = \theta_0 + \frac{\theta_F - \theta_0}{T^3} t^2(3T - 2t)$$

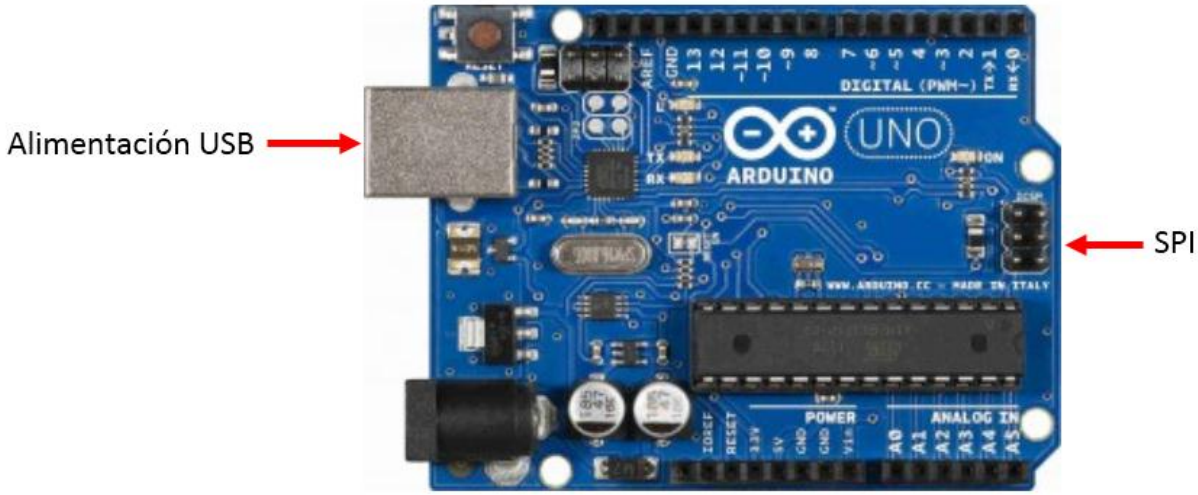
2. Arduino

2.1. Características de Arduino UNO

Arduino es una plataforma de software libre basada en la filosofía del uso sencillo del hardware y del software. Consiste en una placa con circuitería cuyo microcontrolador puede ser programado y en una plataforma de software llamada Arduino IDE (Entorno de desarrollo integrado) que permite cargar códigos desarrollados desde un ordenador.

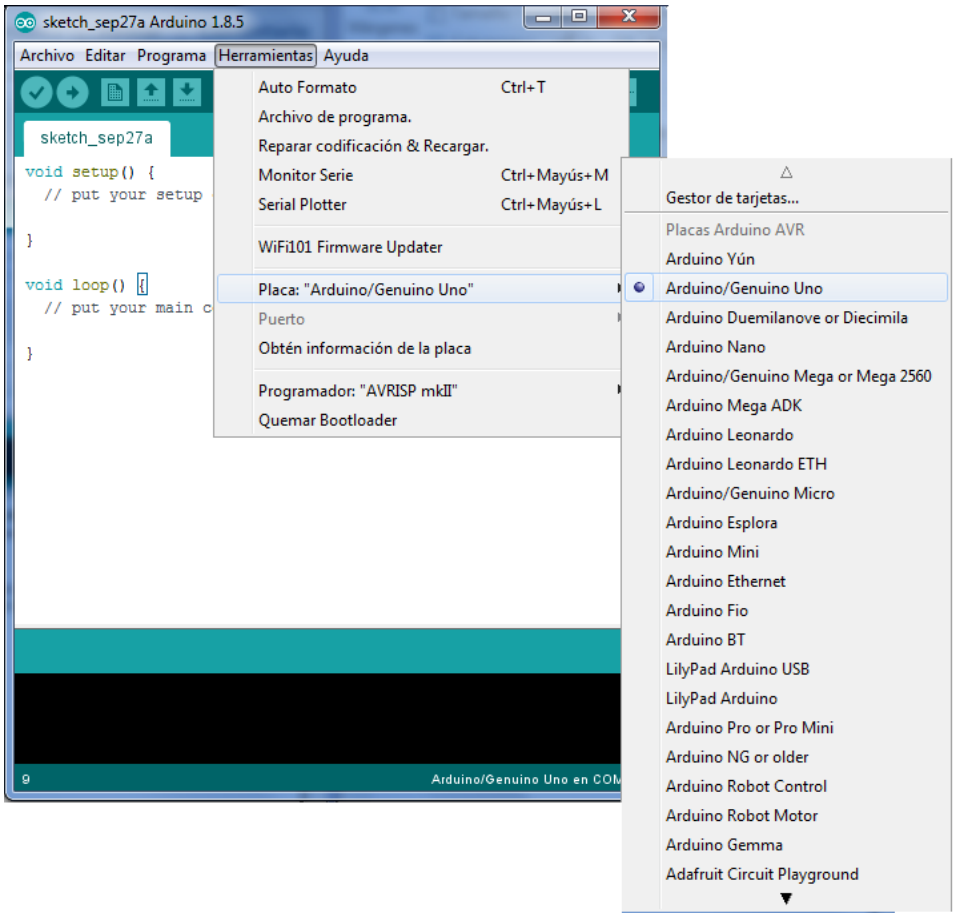
En esta práctica se trabajará con una placa de Arduino UNO y será necesario conocer los elementos siguientes:

- **Alimentación USB:** Arduino puede alimentarse conectándose a un ordenador empleando un cable **USB** tipo B. Desde este puerto se cargarán los programas y se recibirá información desde la placa para su posterior procesado.
- **Pin ICSP**: a través de este pin se realizará la conexión entre la placa Arduino UNO y el sensor Pixy (tratándose éste como un elemento periférico).

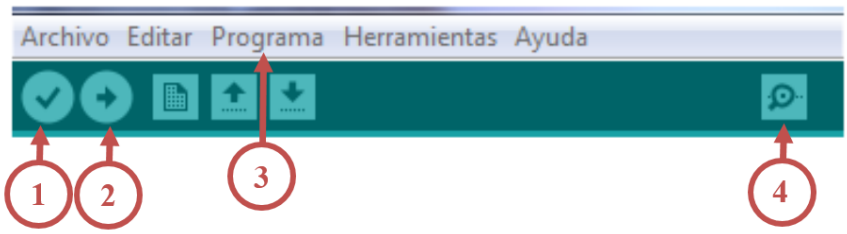


2.2. Entorno de Arduino IDE

Para poder trabajar con Arduino IDE es necesario instalar la aplicación de escritorio de Arduino. Una vez instalada e iniciada la aplicación de escritorio, y tras conectar la placa de Arduino al ordenador, es necesario seleccionar la placa de Arduino para evitar errores durante la carga del programa a la placa. Para ello accedemos al menú **Herramientas** de la barra de herramientas, y seleccionamos Arduino UNO en la opción **Placa**, tal y como puede observarse en la figura siguiente:



En la figura siguiente puede observarse esta barra de herramientas.



Donde:

5. Ejecuta un corrector que permite verificar si existe algún error en el código.
6. Permite cargar el programa desarrollado en la placa de Arduino.
7. Contiene ciertas funciones relacionada con el código, entre ellas la opción de incluir librerías en el paquete de librerías de Arduino. Esto será de utilidad puesto que el sensor Pixy cuenta con su propia librería .zip que será necesario cargar previamente en la aplicación de Arduino.
8. Despliega una ventana donde se muestra la información enviada por el puerto serial una vez se ejecuta el programa en caso de que se hubiese escrito la orden en el código cargado.

2.3. Nociones básicas de programación en Arduino IDE

La estructura básica de un programa de Arduino consta de dos partes en su forma más simple. Estas dos partes son indispensables para su ejecución y en ellas contienen las instrucciones y declaraciones. Estas dos partes se conocen como **setup** y **loop**.

En **setup** se recoge la configuración del programa y es la primera función que se ejecuta en el programa. Esta función sólo se ejecuta una única vez y debe incluirse aunque en ella no se declare nada. En este bloque pueden declararse variables, inicial las comunicaciones seriales, etc.

En **loop** se encuentra contenido el grueso del programa a ejecutar. Tal y como su nombre indica, se trata de un bucle que se repite cíclicamente y sin pausa.

Dos elementos fundamentales del lenguaje de Arduino son las **llaves** {} y el **punto y coma** ;. Las llaves sirven para definir el principio y el final de un bloque de instrucciones, empleándose en funciones, bucles, condiciones, etc. El punto y coma se emplea para separar instrucciones y debe emplearse al finalizar cada instrucción.

```
void setup()
{
  instrucciones;
}

void loop()
{
  instrucciones;
}
```

Una **función** es un bloque de código que se ejecuta cuando se llama a la función. Las funciones se declaran según el tipo de valor "**type**" que la función devuelva. Si la función devuelve un valor entero se iniciarán con **int**¸ mientras que si no se devuelve ningún valor, se iniciará con **void**.

Para construir una función bastaría con iniciar con el tipo de valor, seguido del nombre y acabando con los parámetros de entrada entre paréntesis.

```
type nombre(parámetros)
{
  instrucciones;
}
```

En arduino las **variables** deben declararse previo a su uso y de acuerdo al tipo de información que almacenarán.

```
type nombre;
```

Las variables cuentan con otra propiedad añadida, en función del lugar en el que se declare la variable, ésta podrá ser o no visible para ciertas partes del programa. Una variable **global** es aquella que puede ser vista por cualquier función e instrucción del programa y para ello debe declararse antes del **setup**, al comienzo del programa. Una variable **local** es aquella que se define dentro de un bloque de instrucciones y sólo es visible para ese bloque concreto y para ello debe declararse dentro de ese bloque. En la tabla siguiente se recogen los tipos utilizados con la clase de información que almacenan.

type	Tipo de datos
int	Número entero de 16 bits
int32_t	Número entero de 32 bits
double	Punto flotante de doble precisión en 32 bits
boolean	Valor binario: 0/1 o true/false

Un **array** es un vector al que se accede a cada componente con un número de índice, estando el primer valor bajo el índice 0. Un array se declara escribiendo el tipo de datos que va a almacenar, el nombre, el número de componentes que tendrá entre corchetes, un símbolo = y finalizando con los valores entre llaves y separados por comas.

Type *nombre*[tamaño] = {valor0, valor1, …, valorN};

El bloque de instrucciones **if… else** es en el que se ejecuta la primera parte si se cumple una condición o la se ejecuta la segunda si no se cumple la condición.

```
if (condición)
{
    instrucciones 1;
}
else
{
    instrucciones 2;
}
```

El bloque de instrucciones **for** permite repetir las instrucciones en un bucle. Este bucle se repite desde un valor inicial hasta otro final, variando este valor según una expresión concreta.

```
for (int j = 0; j<10; j++)
{
  instrucciones;
}
```

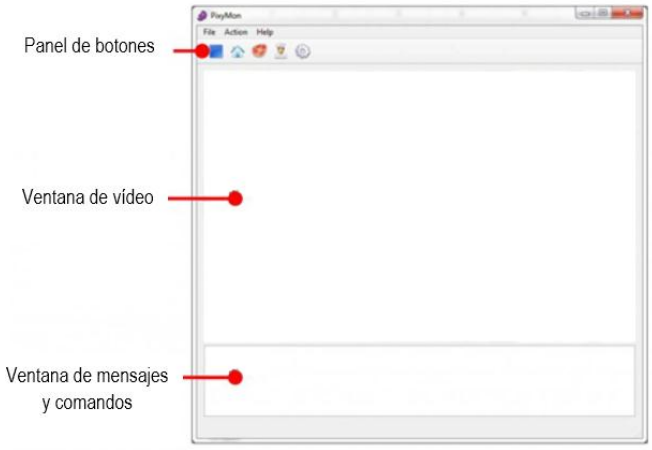
3. Pixy y PixyMon

Pixy es un sensor de visión rápida ideado para aplicaciones de robótica. Este sensor es capaz de almacenar información de hasta 7 colores diferentes y de detectarlos dentro de su campo de visión. Es capaz de detectar y seguir múltiples blancos de forma simultánea y de proporcionar información de la posición de cada objeto en tiempo real. Pixy además cuenta con un sistema pan/tilt que le permite orientarse usando dos servomotores que le provee de dos grados de libertad y un amplio campo visual. Pixy tiene la capacidad de enviar órdenes directas a los servomotores para modificar su posición de forma rápida y eficaz. Además, es capaz de cargar programas desde microcontroladores para realizar actividades de seguimiento y posicionado de objetivos

El sensor Pixy cuenta con una aplicación de escritorio llamada PixyMon que permite interactuar directamente con el sensor Pixy únicamente conectándolo a un ordenador a través de una conexión USB. Esta aplicación permite ver lo que el sensor Pixy capta, ejecutar programas y modificar parámetros de configuración de Pixy.

3.1. Interfaz principal

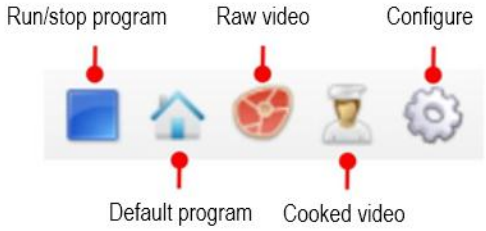
Una vez instalada la aplicación se puede observar la interfaz siguiente:



Donde se distinguen los elementos siguientes:

- **Panel de botones**: barra de herramientas más comunes.
- **Ventana de vídeo**: ventana donde se muestra el vídeo captado por el sensor Pixy
- **Ventana de mensajes y comandos**: ventana desde la que Pixy notifica ciertos mensajes y desde donde se pueden introducir comandos.

El panel de botones cuenta con los elementos siguientes:



Donde cada botón realiza las funciones siguientes:

- **Run/stop program**: este botón permite detener la transmisión de vídeo a través de la ventana de vídeo y de detener o reanudar el programa que se esté ejecutando.
- **Default program**: ejecuta el programa que se encuentra cargado.
- **Raw video**: muestra el vídeo sin procesar. Resulta útil para ajustar el enfoque y la iluminación.
- **Cooked video**: muestra el vídeo sin procesar con una capa de vídeo procesado. En este modo no se envía información a través de los puertos seriales.
- **Configure**: se abre el menú de configuración de parámetros.

3.2. Ejecutar el pan/tilt demo

La aplicación PixyMon dispone de un programa guardado en la memoria que mueve los servomotores para perseguir los objetos que coincidan con la señal 1. Este programa consiste en una aproximación de controlador PD (aproximado porque no contempla el tiempo, sino únicamente las variaciones en la posición entre iteraciones) que busca ubicar el centro del objeto detectado en el centro de la imagen.

Para ejecutar el programa pulsamos la opción **Run the pan/tilt demo** dentro de **Action** en la barra de herramientas.

3.3. Configuración de Pixy en PixyMon

En esta subsección se resumirán los distintos menús de configuración de Pixy en la aplicación PixyMon. La configuración cuenta con 8 menús de configuración distintos dentro de la opción **Pixy Parameters**.

- **Signature Tuning**

  Tal y como se desarrolló en las subsecciones anteriores, esté menú de configuración permite modificar el umbral de inclusión de las señales almacenadas en memoria, modificar el brillo mínimo aceptable y el brillo de la cámara

- **Signature Labels**

  Este menú de configuración permite modificar el nombre que muestran las señales almacenadas en el vídeo al ser detectadas.

- **Expert**

  Este menú de configuración contiene parámetros avanzados del sensor Pixy. Se recomienda no modificarlos.

- **Blocks**

Cada objeto que el sensor Pixy detecta es catalogado como "block" que lleva asociado una serie de parámetros que permiten identificar a qué señal corresponde, su tamaño y ubicación en la imagen. Este menú de configuración permite modificar el número máximo de "blocks" que se permiten, el máximo número de "blocks" por cada señal el área mínima (ancho·alto) a partir del cual se acepta la detección como un "block"

- **Interface**

Este menú de configuración contiene los parámetros de comunicación de Pixy hacia el exterior. Es posible seleccionar el puerto de salida de datos (Arduino IC SP SPI, I2C, UART,…), la dirección del puerto de salida I2C y la tasa de baudios del Transmisor-Receptor Asíncrono Universal (UART).

- **Camera**

Este menú de configuración contiene parámetros acerca de cómo el sensor Pixy adquiere imágenes para su procesado. Se recomienda no modificarlos.

- **Servo**

Este menú de configuración contiene parámetros de configuración de los puertos en el sensor Pixy de los dos servomotores. Los límites superior e inferior representan como de lejos se puede mover el servo en sentido antihorario y en sentido horario respectivamente.

- **Pan/tilt Demo**

Este menú de configuración contiene los parámetros que permiten ajustar el controlador de persecución. Es posible modificar las ganancias P y D asociadas al giro vertical y horizontal del controlador "PD"

4. Puesta a punto de la práctica

4.1. Material facilitado

Para el desarrollo de la práctica se proporcionará el siguiente material:

- Montaje del sensor Pixy y placa de Arduino UNO.
- Conector USB de tipo mini B.
- Conector USB de tipo B.
- Panel doble con bisagra.
- Objetos detectables.
- Archivos de Arduino:
    - rastreo.ino
    - trayectorias.ino
    - Librerias de Pixy.
- Archivos de MATLAB:
    - Interfaz_Pixy.m
    - Interfaz_Pixy.fig
    - Interfaz_vision.m
    - Interfaz_vision.fig

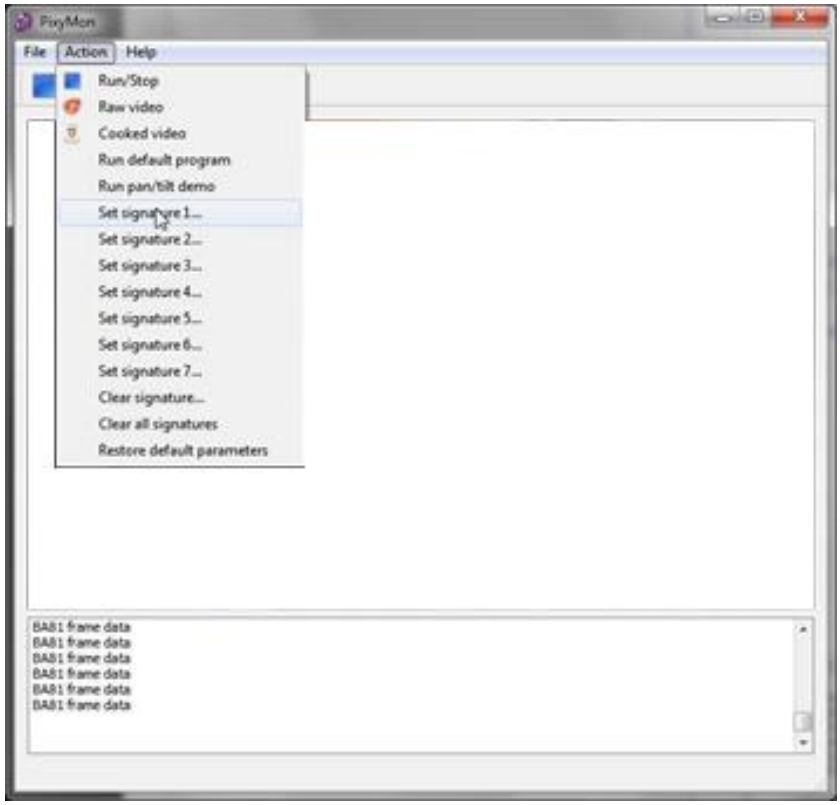
- Utilidad CoolTerm.
- Ejecutables de PixyMon y de Arduino IDE.

Previamente es necesario instalar los ejecutables de PixyMon y de Arduino para el desarrollo de la práctica.

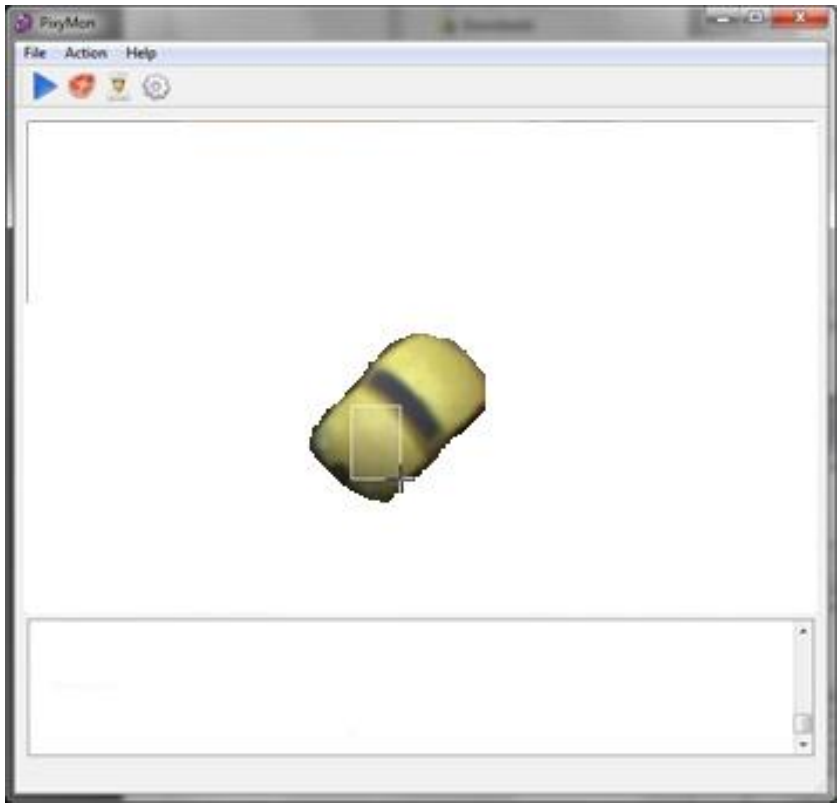
4.2. Aprendizaje de señales

La primera parte de la práctica consistirá en enseñar los elementos que deseemos detectar a Pixy. Para ello conectaremos el sensor al ordenador usando el cable USB de tipo mini B. Realizaremos el proceso siguiente para las distintas señales que se quieran guardar:

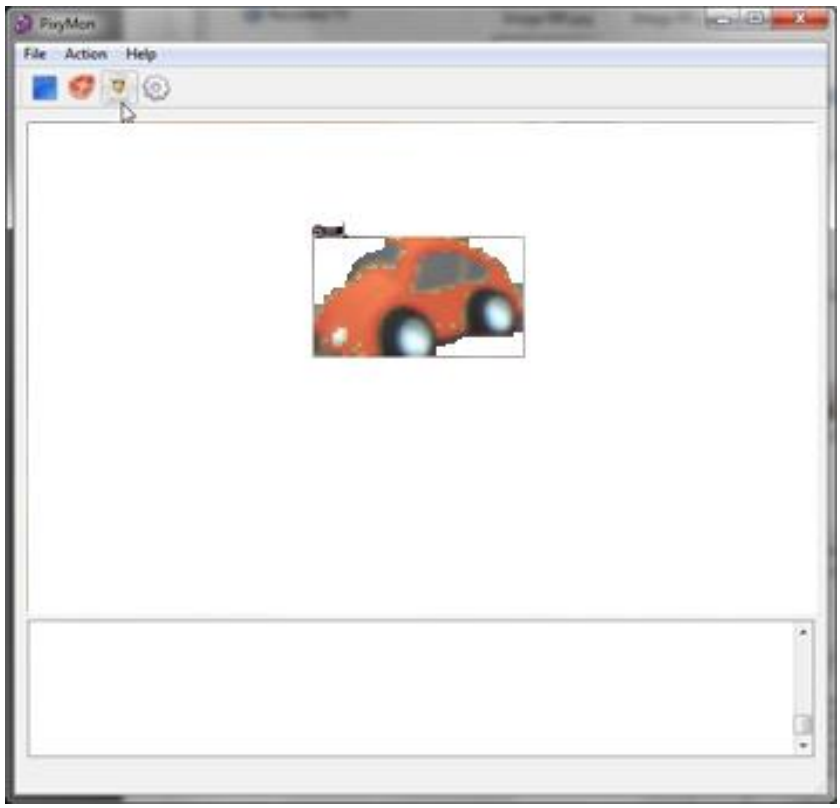
1. Una vez abierta la interfaz principal de PixyMon, activar la opción **Raw video** y mantener el objeto que se quiere enseñar a Pixy y seleccionar la opción **Set signature X** dentro de la opción **Action** de la barra de herramientas, tal y como se muestra en la figura siguiente:



2. Una vez seleccionada la señal en la que se desea guardar la información el vídeo parará de refrescarse y podremos seleccionar la región a analizar. Para ello, pulsar y mantener el botón izquierdo del ratón y arrastrar hasta abarcar el área deseada, tal y como se muestra en la figura siguiente:



3. Una vez realizado, pulsar el botón **Cooked video** del panel de botones para observar cómo el sensor Pixy detecta los píxeles del color guardado. En la figura siguiente puede observarse el resultado:



Es necesario remarcar que las señales guardadas se almacenan en una memoria flash, por lo tanto, al cesar el suministro de energía eléctrica la información de las señales se perderá.

Una vez aprendidas las señales es posible activar el modo **Pan/tilt demo** para observar cómo el sensor Pixy persigue elementos

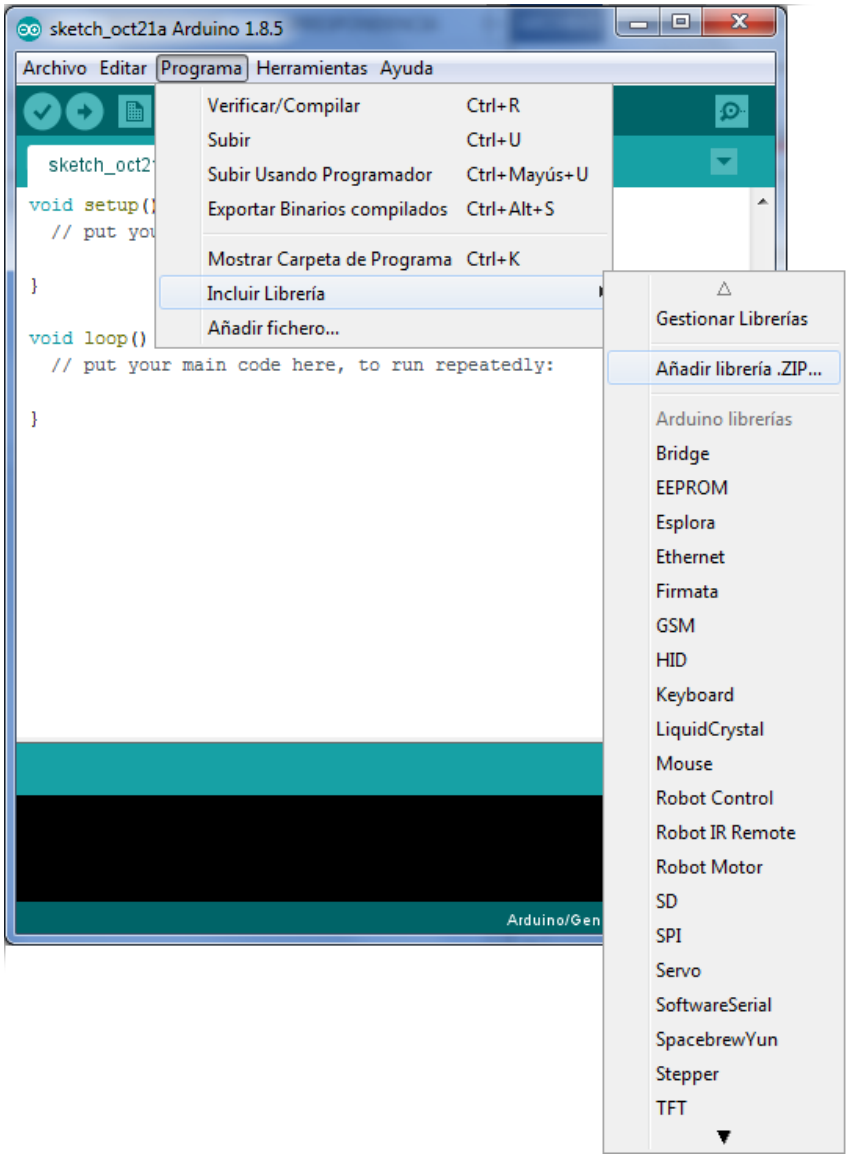
4.3. Montaje del fondo y posición de elementos

Una vez guardadas las señales de los elementos que van a emplearse es necesario ubicar el fondo y posicionar los elementos dentro del campo visual del sensor Pixy. Es recomendable colocar el fondo a una distancia de 40 cm y con una apertura de 100º.

5. Detección de elementos

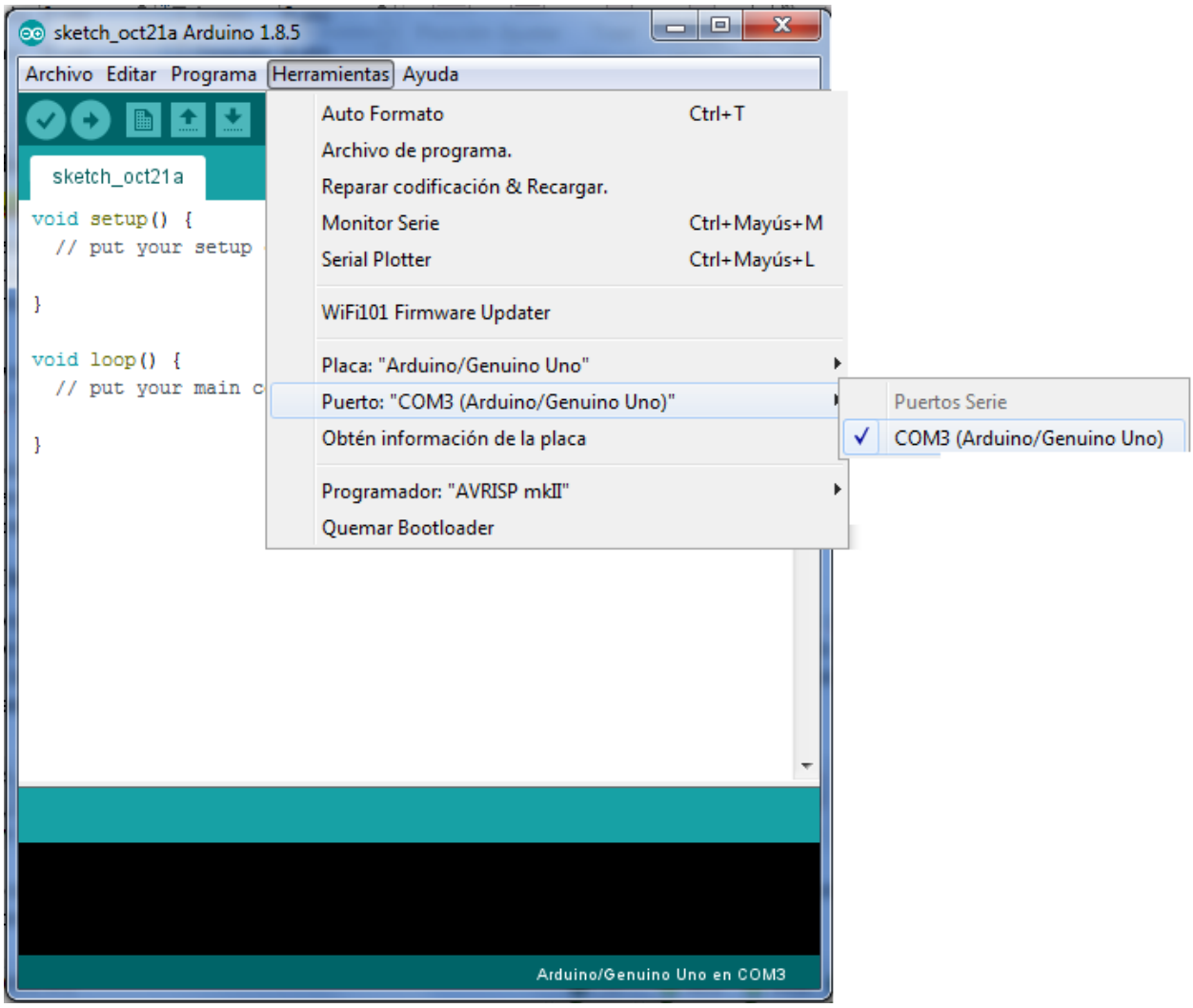
5.1. Arduino

Una vez realizado el posicionamiento de los elementos y grabado en la memoria del sensor Pixy las señales, es necesario cargar las librerías propias de Pixy en el programa de Arduino IDE. Las librerías se encuentran en el archivo comprimido proporcionado y su nombre es **arduino_Pixy-0.1.7.zip**. Para incluir las librerías bastaría entrar en la opción Programa, incluir librería y finalmente añadir librería .ZIP, tal y como podemos observar en la imagen siguiente.



Este paso sólo debe realizarse una única vez una vez instalada la aplicación de Arduino.

El siguiente paso sería conectar la placa de Arduino IDE a través del puerto USB y configurar la aplicación de Arduino para reconocer el **puerto serial** y la **placa de Ardunio UNO**.



Es necesario remarcar que al conectar la placa de Arduino al ordenador, es posible que envíe información al sensor Pixy del programa que tuviera la última vez antes de cortar la alimentación, por ello se recomienda cargar un programa vació antes de cargar cualquier otro y **parar la ejecución del programa por defecto desde PixyMon**.

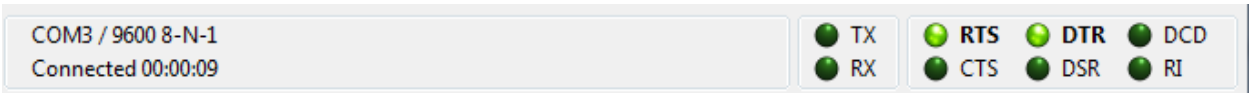
Una vez pausada la ejecución del programa por defecto desde PixyMon abriríamos el archivo de Arduino **rastreo.ino** y cargaríamos el programa empleando el botón **subir**, tal y como se indica en la guía rápida de Arduino.

Se recuerda la importancia de pausar la ejecución en PixyMon de la ejecución del programa por defecto, ya que es necesario la realización de otro paso antes de ejecutar el programa. En caso de error, bastaría pausarlo y cargar de nuevo el programa.

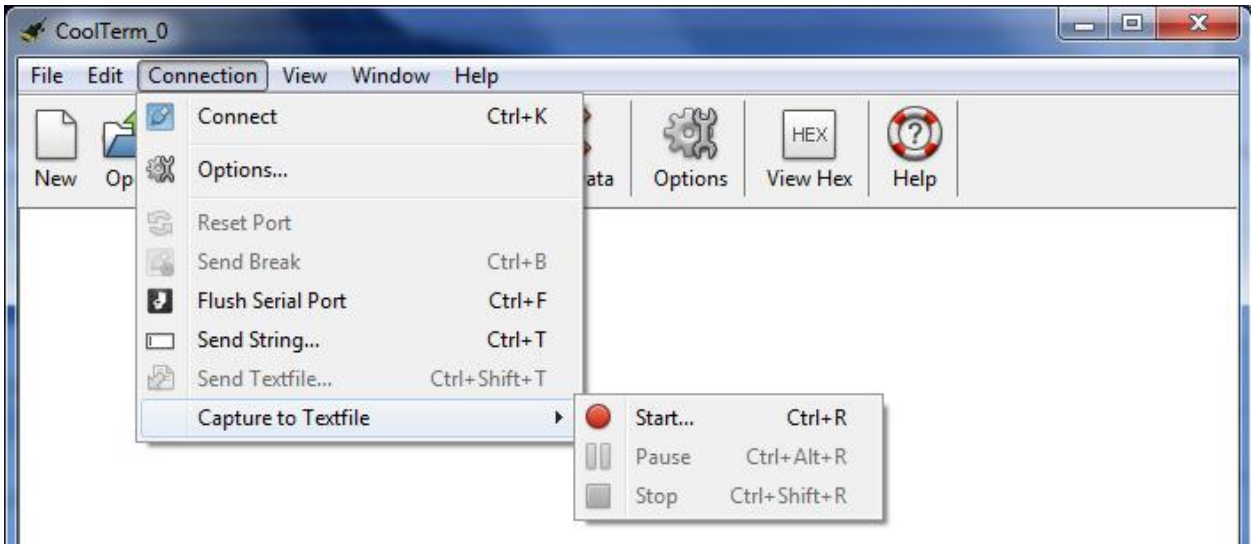
5.2. Inicio de CoolTerm

Este ejecutable permite almacenar la información recabada por el sensor Pixy en un archivo .txt para su pposterior procesado. Por lo tanto, una vez cargado el programa, se conectará este programa al puerto serial y se almacenará la información recibida. Para conectar la aplicación CoolTerm al puerto serial es necesario pulsar la opción **conectar** de la barra de herramientas de la aplicación.

Para confirmar que se ha establecido la conexión con el puerto serial se deben iluminar los botones de la barra inferior de la aplicación de forma similar a la que se puede observar en la siguiente figura.

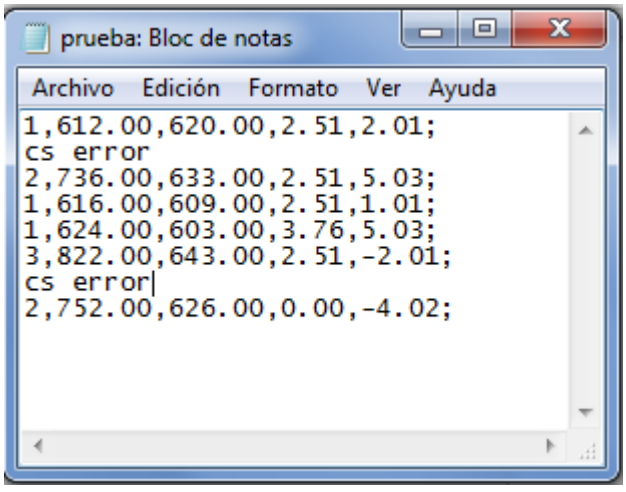


Para **grabar la información** en un archivo .txt es necesario crear primero ese mismo archivo desde la aplicación, tal y como se puede observar en la figura siguiente.



Los datos se almacenarán en un archivo de texto con la siguiente organización. Cada detección realizada se ubicará en una fila distinta y consecutiva a la anterior. La información de cada detección será la siguiente: número de la señal, posición del servomotor horizontal, posición del servomotor vertical, error horizontal y error vertical. La información estará separada por comas y la línea acabará con un punto y coma. Cabe destacar que el puerto serial puede enviar información errónea que se traduce en una línea con el texto *cs error* escrito.

En la figura siguiente puede observarse un ejemplo de información obtenida con esta metodología.



5.3. Ejecución desde PixyMon

Una vez realizado este proceso bastaría con **activar la ejecución del programa por defecto** y observar cómo el sensor Pixy realiza un barrido y recoge la información de los elementos detectados.

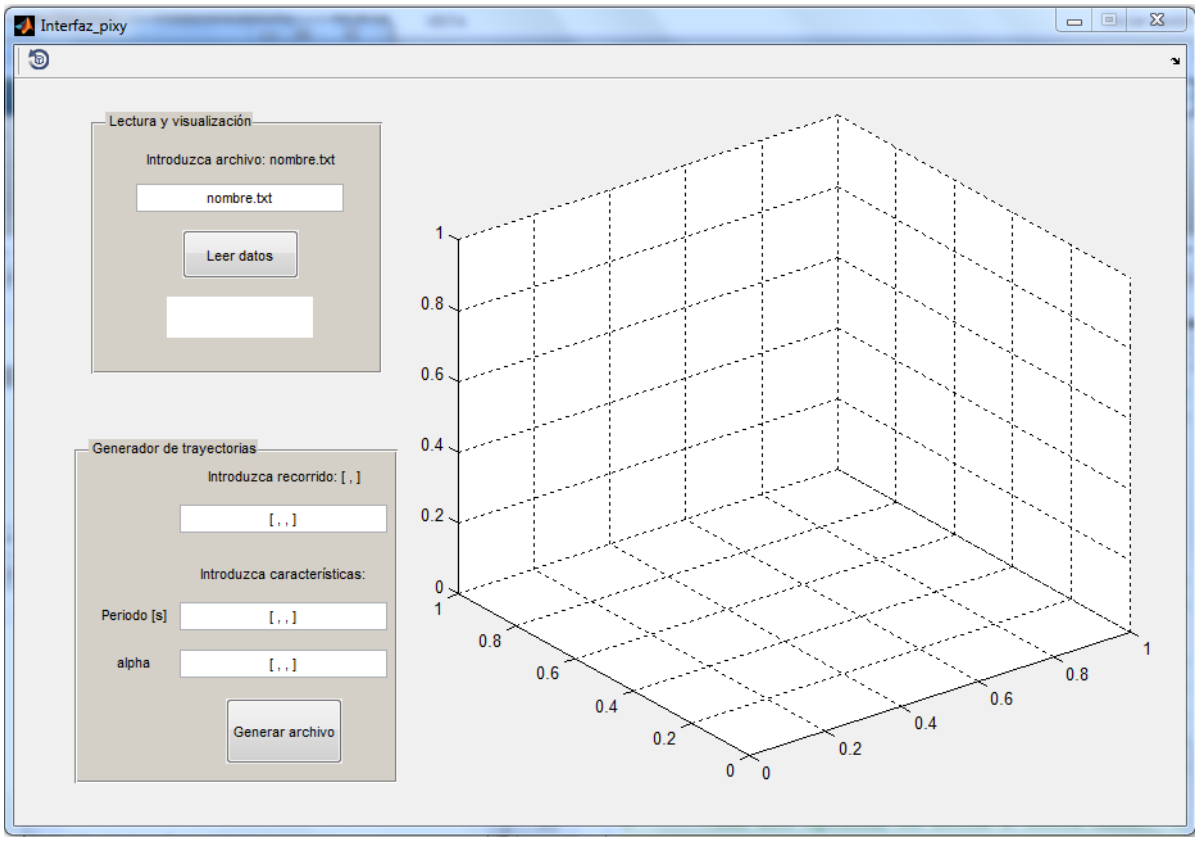
6. Procesado de datos en la interfaz de usuario de MATLAB

Esta interfaz permite al alumno procesar la información obtenida del rastreo y generar un archivo en Arduino que ejecute las trayectorias deseadas.

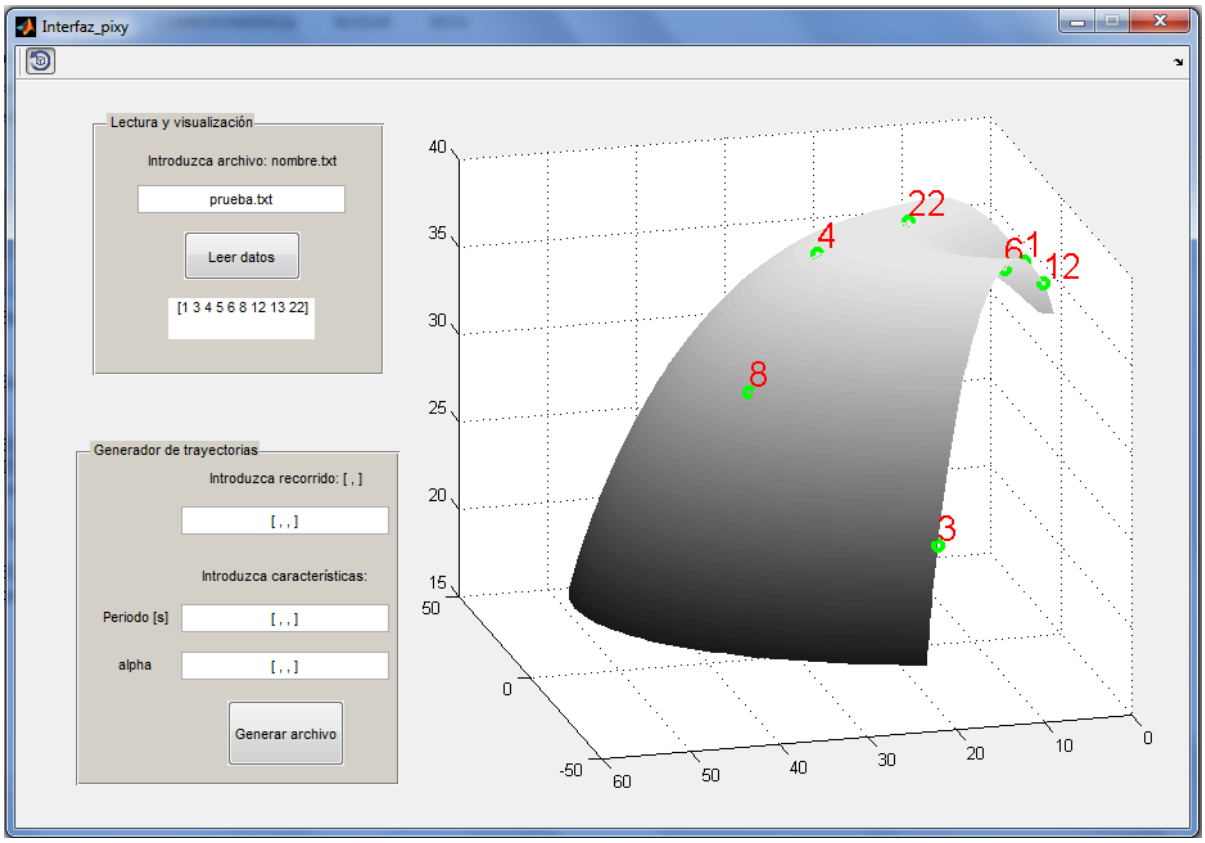
Esta interfaz cuenta con tres bloques diferenciados que se explican a continuación:

4. **Lectura y visualización**: este bloque permite leer y cargar la información obtenida del algoritmo de detección de elementos. El pulsador *Leer datos* permite cargar la información y representarlas en el gráfico.
5. **Generador de trayectorias**: Permite introducir el orden del recorrido a realizar con las características en tiempo y par deseadas. Además genera un archivo con el código necesario para ejecutar en Arduino IDE.
6. **Gráfico**: Representa el campo visual del sensor Pixy y la ubicación de los elementos en el mismo.

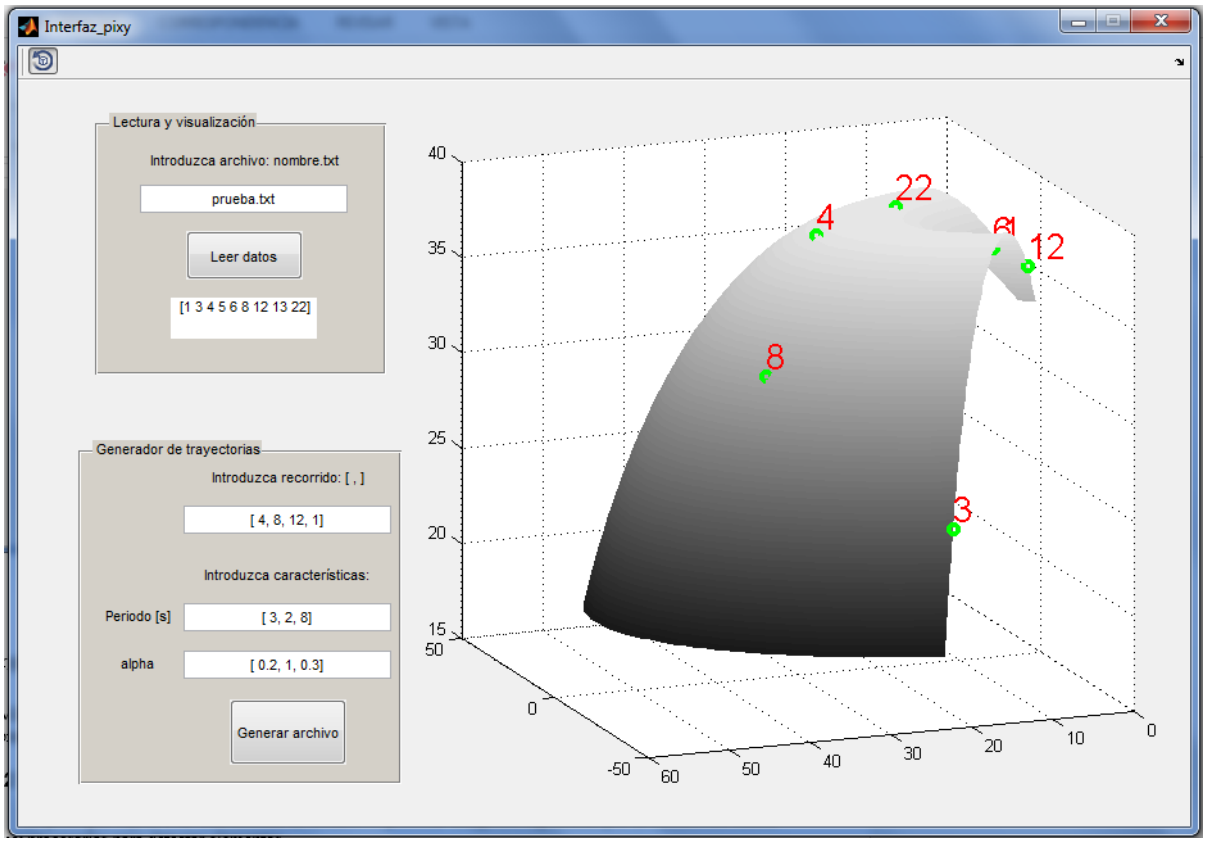
En la figura siguiente puede observarse el entorno de la interfaz gráfica inicial.



En la figura siguiente puede observarse el entorno gráfico con la información cargada y representada.



En la figura siguiente puede observarse un ejemplo de introducción de trayectorias.



La trayectoria por defecto es la de tiempo mínimo, por lo que para establecer una trayectoria de mínima energía se ha definido para un valor del parámetro $\alpha = 1$.

7. Ejecución de las maniobras

Una vez obtenido el archivo de trayectorias de la interfaz de usuario bastaría con **abrir el archivo en Arduino, cargar el programa en la placa de Arduino** y **activar la ejecución del programa por defecto**.

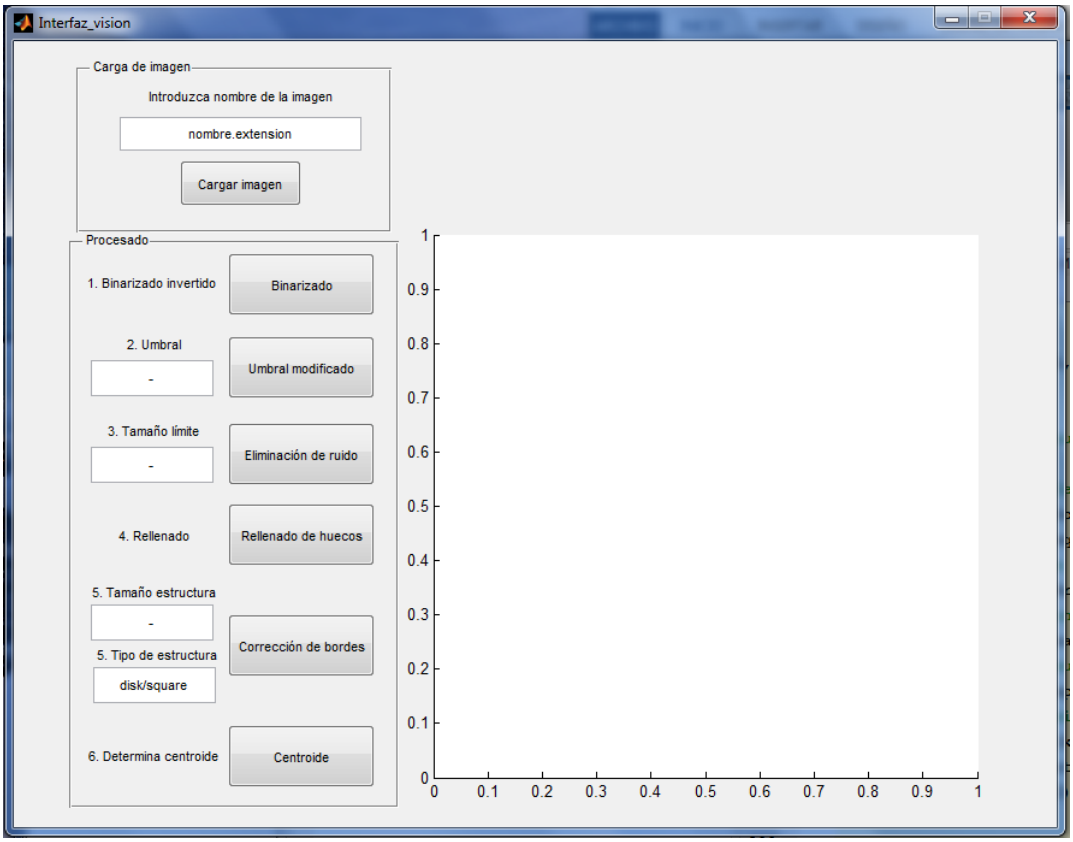
8. Procesado de imágenes en la interfaz de usuario de MATLAB

El principal objetivo de esta interfaz de usuario es procesar imágenes capturadas por la cámara del sensor Pixy y procesarlas para detectar elementos.

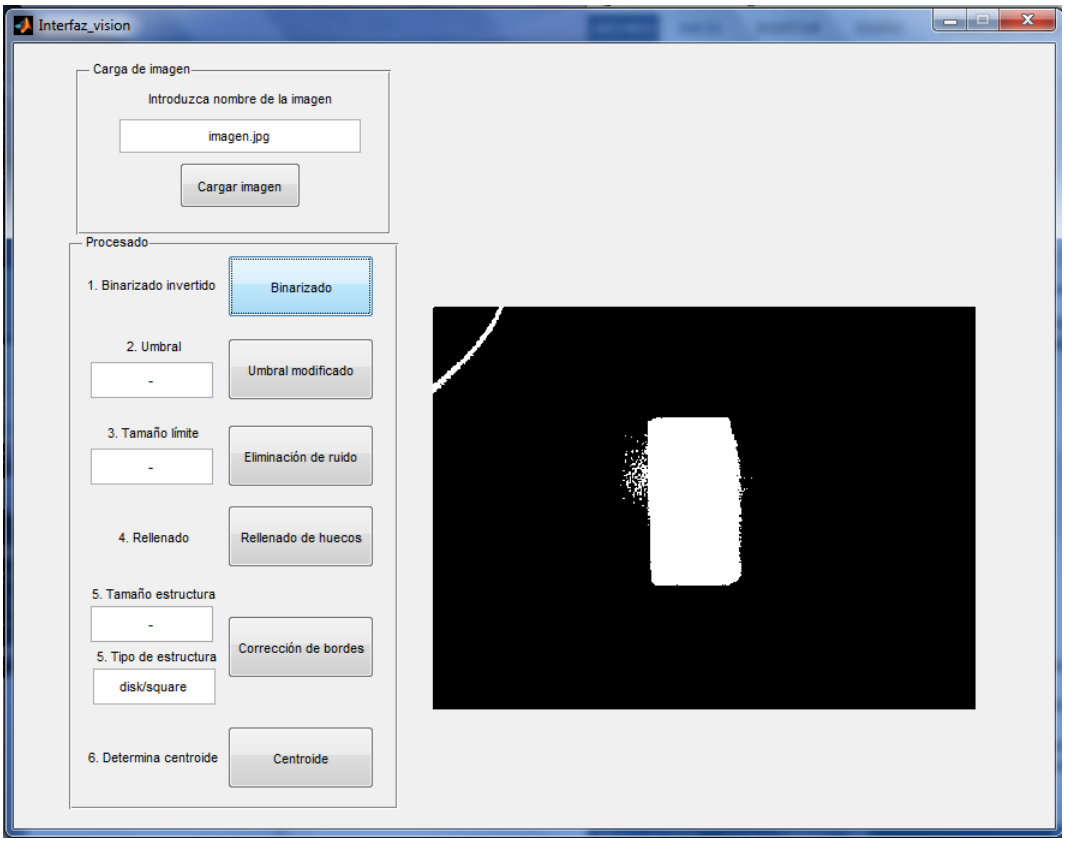
Esta interfaz cuenta con dos bloques diferenciados que se explican a continuación:

1. **Carga de imagen**: este bloque permite la carga y representación de la imagen a procesar.
2. **Procesado**: este bloque permite el procesado de la imagen con los siguientes pasos: binarizado de la imagen, modificación del umbral, eliminación del ruido, rellenado de huecos, corrección de los bordes según la forma y posicionado el centroide el elemento.
3. **Gráfico**: este bloque muestra la imagen tratada.

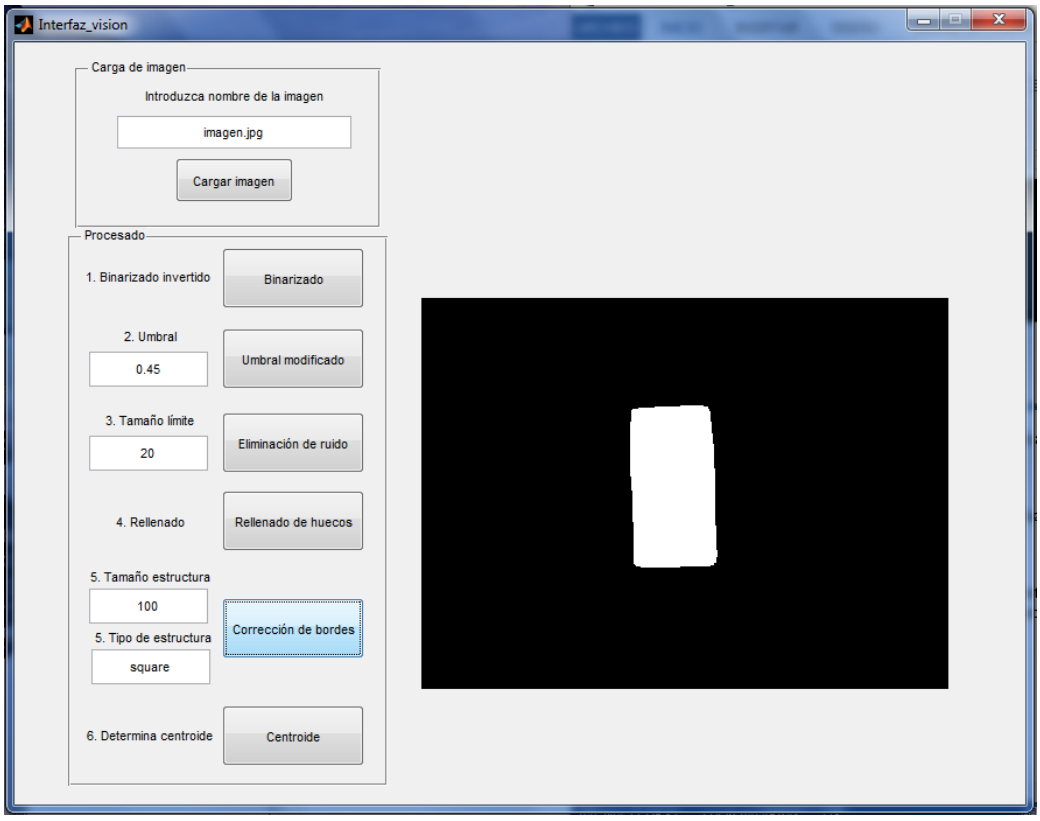
En la figura siguiente se puede observar la interfaz de visión inicial.



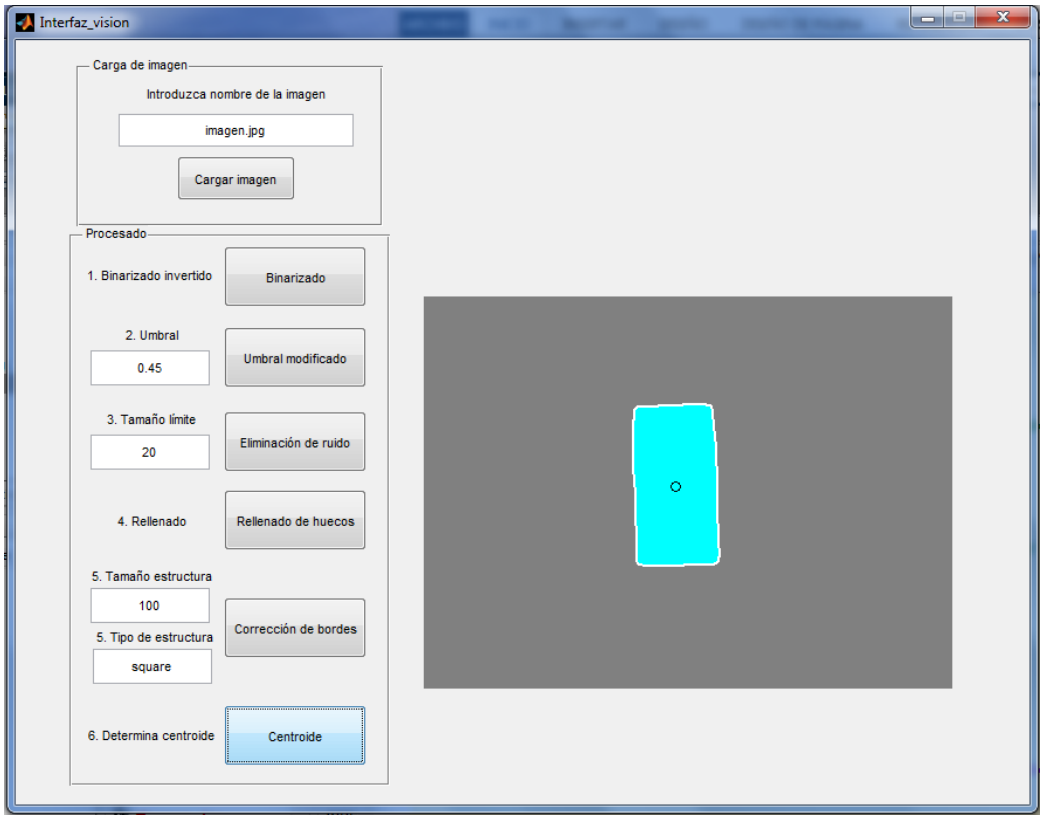
En la figura siguiente se puede observar un ejemplo de imagen con binarizado invertido. Dado que las funciones posteriores trabajan con el color blanco, se ha optado por representar como 1 - binarizado.



En la figura siguiente se puede observar el resultado de aplicar los pasos 2 a 5.



En la figura siguiente se puede observar el resultado final del procesado.



El umbral es un valor entre 0 y 1 que asigna el color blanco o negro tomando como punto de corte este umbral. Los valores tamaño límite y tamaño estructura son tamaños de píxeles. En el primer caso el tamaño límite a partir del cual se eliminan elementos y el segundo caso el tamaño de la estructura de referencia para suavizar los bordes.

APÉNDICE B. ARCHIVOS DE ARDUINO IDE

Algoritmo 1: *posiciona.ino*

Este algoritmo permita posicionar los servomotores en una posición fija.

```
#include <SPI.h>
#include <Pixy.h>
Pixy Pixy;

void setup()
{
  Pixy.init();
}

void loop()
{
  Pixy.setServos(500,500);
  delay(1000);
}
```

Algoritmo 2: *rastreo.ino*

Este algoritmo permite realizar el rastreo y detección de elementos.

```
#include <SPI.h>
#include <Pixy.h>
Pixy Pixy;

#define X_CENTER        ((PIXY_MAX_X-PIXY_MIN_X)/2)
#define Y_CENTER        ((PIXY_MAX_Y-PIXY_MIN_Y)/2)

double dt;                      // diferencial de tiempo.
int step_pan = 250;             // incremento en la posición horizontal.
int step_tilt = 250;            // incremento en la posición vertical.
int32_t pan_tilt_pos[2];        // vector que almacenará la posición de los servos para la trayectoria de barrido.
boolean cont = true;            // variable booleana que permite fijar la posición final.
```

```
double error_prev_tilt = 0;     // error vertical en la iteración anterior.
double error_prev_pan  = 0;     // error horizontal en la iteración anterior.
double error_ac_tilt   = 0;     // integral vertical acumulada en la iteración anterior.
double error_ac_pan    = 0;     // integral horizontal acumulada en la iteración anterior.
int32_t pos_pan;                // posición horizontal inicial.
int32_t pos_tilt;               // posición vertical inicial.

//////////////////////////// PID horizontal ////////////////////////////////////

int panPID(int error)
{
  double PID; // Incremento en la posición por el controlador.
  dt = millis()/1000 - dt/1000; // Diferencial de tiempo.
// Controlador PID.
  PID = (0.1*error - 0.0003*(error - error_prev_pan)/dt  + 0.001*(error_ac_pan + error*dt));
  pos_pan += PID; // Posición del servo.

  if (pos_pan>PIXY_RCS_MAX_POS)
  {
// En caso de que el controlador exija una posición superior al máximo.
     pos_pan = PIXY_RCS_MAX_POS;
  }
    else if (pos_pan<PIXY_RCS_MIN_POS)
    {
      pos_pan = PIXY_RCS_MIN_POS; // En caso de que el controlador exija una posición inferior al mínimo.
    }
  return pos_pan; // Devuelve la posición del servo

  error_ac_pan   = error_ac_pan + error*dt;  // Calcula la integral para la iteración posterior.
  error_prev_pan = error;                    // Error para la iteración posterior.
}

//////////////////////////// PID Vertical ////////////////////////////////////

int tiltPID(int error)
{
  double PID;
  dt = millis()/1000 - dt/1000;
```

```
  PID = (0.1*error - 0.0002*(error - error_prev_tilt)/dt  + 0.001*(error_ac_tilt + error*dt));
  pos_tilt += PID;

  if (pos_tilt>PIXY_RCS_MAX_POS)
  {
     pos_tilt = PIXY_RCS_MAX_POS;
  }
   else if (pos_tilt<PIXY_RCS_MIN_POS)
   {
     pos_tilt = PIXY_RCS_MIN_POS;
   }

  return pos_tilt;

  error_ac_tilt   = error_ac_tilt + error*dt;
  error_prev_tilt = error;
}

////////////////////////////// Trayectoria de rastreo ////////////////////////////////////

int rastreo_pan_tilt()
{
  pan_tilt_pos[1] += step_tilt; // Se incrementa la posición vertical para barrer en vertical.

  if (pan_tilt_pos[1] > 750)    // En caso de encontrarnos en la última posición de un barrido.
  {
  if (pan_tilt_pos[0] == 1000)  // En caso de encontrarnos en la última posición del último barrido vertical.
  {
  cont = false;
  pan_tilt_pos[0] = 1000;
  pan_tilt_pos[1] = 750;
  }
  else
  {
  pan_tilt_pos[0] += step_pan;
  pan_tilt_pos[1] = 250;
  }
  }
```

```
}

/////////////////////////// Inicialización ////////////////////////////////////////

void setup()
{
  Serial.begin(9600);       // Se inicializa el puerto serial.

  Pixy.init();              // Se inicializa Pixy.
  dt = millis()/1000;       // Se inicializa la cuenta del tiempo para la primera iteración.
  pan_tilt_pos[0] = 0;      // Se fija la primera posición horizontal.
  pan_tilt_pos[1] = 250;    // Se fija la primera posición vertical.
}

///////////////////////////// Bucle principal ////////////////////////////////////////

void loop()
{
  int blocks;                           // Variable que almacena el número de elementos detectados.
  double panPOS, tiltPOS, panError, tiltError; // Variables de posición y error empleadas en el PID.
  int flag = 0;                         // Variable que discriminará errores de detección.
  int block_max = 0;                    // Variable que permitirá almacenar el número de elementos detectados.

  ////////////////////////// Posición de rastreo ////////////////////////////////

  rastreo_pan_tilt();                   // Ejecución de la función de rastreo. Devuelve las posiciones de cada
iteración.
  Pixy.setServos(pan_tilt_pos[0],pan_tilt_pos[1]);  // Posiciona los servos en los valores de la trayectoria.
  delay(500);

  ////////////////////////// Inicialización del PID ////////////////////////////////

  error_prev_tilt = 0;
  error_prev_pan  = 0;
  error_ac_tilt   = 0;
  error_ac_pan    = 0;
  pos_pan  = pan_tilt_pos[0];   // Se comienza por el valor de la posición de rastreo.
  pos_tilt = pan_tilt_pos[1];   // Se comienza por el valor de la posición de rastreo.
```

```
 ////////////////////////// Bucle de detección ////////////////////////////
 // Se realiza varias veces por problema de refresco, falsos negativos y falsos positivos.
 for (int j = 0; j<120; j++)
 {
  delay(20);
  blocks = Pixy.getBlocks();

  if (blocks > block_max)
  {
  block_max = blocks;
  }
 }

 ////////////////////////// Vector de tipo de señales ////////////////////////////

 int signature_it[block_max]; // Creamos el vector de almacenamiento de tipo de señales

 for (int t = 0; t<block_max; t++)
 {
  signature_it[t] = Pixy.blocks[t].signature; // Almacenamos el tipo de señal de cada objeto.
 }

 ////////////////////////// Bucle de posicionamiento de objetivos ////////////////////////

// < porque empezamos en 0 los índices. Realizamos un PID por cada elemento detectado.
 for (int k = 0; k<=block_max-1; k++)
 {

   int sign = signature_it[k]; // Creamos una variable con la señal de cada iteración

 // Al igual que al detectar si hay elementos, se realiza tantas veces para asegurar su correcta detección
     for (int j = 0; j<250; j++)
     {
       ///// Retraso por frecuencia de muestreo de PIXY ////////////////
       delay(20);
       blocks = Pixy.getBlocks();
```

```
      ////// Bucle del PID //////

      if (blocks>0 && cont == true) // Si se realiza la detección se accede al bucle del PID.
      {
       int h; // Permite acceder a la posición correcta.

         for (int w = 0; w < blocks; w++ ) // Se realiza un bucle que barra las señales detectadas.
         {
          int signature = Pixy.blocks[w].signature;

          // Si coincide con la señal del elemento de la iteración, entonces se realiza el PID.
          if (signature == sign)
          {
           h = w; // Emplea la posición adecuada para acceder a la información de la posición

           flag++; // Almacena el número de veces que se realiza el PID.

           panError   = X_CENTER - Pixy.blocks[h].x; // Error horizontal.
           tiltError  = Pixy.blocks[h].y - Y_CENTER; // Error vertical.

           panPOS  = panPID(panError);    // Ejecuta la posición del controlador PID horizontal.
           tiltPOS = tiltPID(tiltError);  // Ejecuta la posición del controlador PID horizontal.

           Pixy.setServos(panPOS, tiltPOS);   // Posiciona los servomotores.
          }
         }
        }
       }

      ////////////////////////// Envío de información al puerto serial ////////////////////////////

       if (flag>40) // Si ha sido detectado suficientes veces se envía la información al puerto serial.
       {


Serial.print(sign);Serial.print(",");
          Serial.print(panPOS);Serial.print(",");
          Serial.print(tiltPOS);Serial.print(",");
```

```
        Serial.print(panError*200/319);Serial.print(",");
        Serial.print(tiltError*200/199);Serial.println(";");
      }
   delay(500);
  // Reinicia a la posición de la trayectoria para cada elemento.
   Pixy.setServos(pan_tilt_pos[0], pan_tilt_pos[1]);
   delay(500);
 }
}
```

Algoritmo 3: *trayectorias.ino*

Este algoritmo permite ejecutar las trayectorias.

```
// Incluimos las librerías necesarias.

#include <SPI.h>
#include <Pixy.h>
Pixy Pixy;

// Definición de las variables.

boolean cont = true;  // Definimos la variable booleana de control que finaliza las trayectorias y para los servos.

int j;                          // Definimos la cuenta de trayectorias (ej: j = 2 implica que tenemos 3 trayetorias (0,1,2)).
double t;                       // Definiimos el tiempo local de cada trayectoria.

int          j_max = 1;         // Porque la cuenta de j empieza en 0.
double    indicador[2] = {0, 1};     // Indicamos si queremos trayectoria de tipo 0 () o de tipo 1 ().
// Valor de alpha de la trayectoria. Sólo se emplea valores si el indicador vale 0. Para indicador 1, usamos 1.
double       alpha[2] = {0.2, 1};
double          T[2] = {5, 10};     // Duración de la trayectoria.
double ang_pan_ini[2]  = {0, 1000};   // Posiciones horizontales iniciales.
double ang_tilt_ini[2] = {500, 500};  // Posiciones verticales iniciales.
double ang_pan_fin[2]  = {1000, 300}; // Posiciones horizontales finales.
double ang_tilt_fin[2] = {500, 1000};  // Posiciones verticales finales.
```

```
// Bucle de inicialización (sólo se ejecuta una vez).

void setup()
{
  Serial.begin(9600); // Inicialización del serial.
  j = 0;              // Comenzamos con la trayectoria 0.
  t = 0;              // Se inicializa el tiempo en 0 para la trayectoria 0.
  Pixy.init();        // Inicialización de Pixy.
}

// Bucle principal.

void loop()
{
  double pan_serv;   // Definimos la posición del servo horizontal para uso en el bucle principal.
  double tilt_serv;  // Definimos la posición del servo vertical para uso en el bucle principal.

  if (t == 0 && j < j_max) // Si nos encontramos al inicio de la PRIMERA trayectoria (t=0).
  {
    Pixy.setServos(ang_pan_ini[j],ang_tilt_ini[j]); // Nos movemos a los inicios de trayectorias.
    delay(3000);
  }

  if (cont == true) // Si la variable booleana permite las trayectorias.
  {


                                                                                    //
MANIOBRA DE TIEMPO MÍNIMO


    if (indicador[j] == 0)
    {
      if (t/T[j] <= alpha[j]) // Primera parte de la trayectoria.
      {
        pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] - ang_pan_ini[j])*pow(t,2)/pow(T[j],2)/alpha[j];
        tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] - ang_tilt_ini[j])*pow(t,2)/pow(T[j],2)/alpha[j];
      }
      if (t/T[j] > alpha[j])  // Segunda parte de la trayectoria.
      {
```

```
      pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] - ang_pan_ini[j])*( alpha[j] + 2*( t/T[j] - alpha[j] ) + pow(( t/T[j] - alpha[j] ),2)/(alpha[j] - 1) );
      tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] - ang_tilt_ini[j])*( alpha[j] + 2*( t/T[j] - alpha[j] ) + pow(( t/T[j] - alpha[j] ),2)/(alpha[j] - 1) );
    }
   }


                                                                                                    //
MANIOBRA DE MÍNIMA ENERGÍA


   if (indicador[j] == 1)
   {
    pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] - ang_pan_ini[j])*(3*T[j] - 2*t)*pow(t,2)/pow(T[j],3);
    tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] - ang_tilt_ini[j])*(3*T[j] - 2*t)*pow(t,2)/pow(T[j],3);
   }


                                                                                                    //
POSICIONADO DE LOS SERVOS


   Pixy.setServos(pan_serv,tilt_serv); // Movemos a las posiciones calculadas
   delay(10);                    // Generamos un retraso de 10 ms
// Aumentamos el tiempo para calcular la posición tras esos 10 ms (es bastante aproximado dado el tiempo de procesamiento)
   t = t + 0.01;
   //Serial.println(t);
   if (t >= T[j]) // Cuando se supere el periodo de la trayectoria se finaliza.
   {
    j++;       // Siguiente trayectoria.
    t = 0;     // Tiempo local puesta a 0
    delay(5000); // Pausa de X ms entre trayectoria y trayectoria.
   }
  }
  if (j > j_max) // Cuando se acabe la última trayectoria se deja de mover los servos.
  {
   cont = false;
  }
}
```

APÉNDICE C. ARCHIVOS DE MATLAB

Script 1: *Interfaz_Pixy.m*

Este script, junto con *Interfaz_Pixy.fig*, genera la interfaz gráfica de trayectorias.

```
function varargout = Interfaz_Pixy(varargin)

clearvars -except varargin;
gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_Pixy_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_Pixy_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function Interfaz_Pixy_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;

guidata(hObject, handles);

function varargout = Interfaz_Pixy_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

%%  Cajones de editables

function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)
```

```
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)

function edit3_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

function edit4_CreateFcn(hObject, eventdata, handles)

if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% Pulsadores %%

function pushbutton1_Callback(hObject, eventdata, handles)

hold off, axes1_CreateFcn(hObject, eventdata, handles);
% Matriz de almacenamiento de los datos
datos  = [];
% Abrimos el archivo de texto, y generamos su identificador
fid    = fopen(get(handles.edit1,'string'));

if fid == -1
   disp(' ');
   disp('¡EL NOMBRE DEL ARCHIVO NO SE ENCUENTRA EN EL DIRECTORIO!');
   disp('Sugerencias: ');
   disp('- Revise si ha incluido el archivo en el directorio del...
ejecutable GUIDE. ')
   disp('- Revise errores en el nombre.');
   disp('- No incluya espacios en el nombre o sustituyalos por _ .');
   disp('- No olvide incluir la extension .txt');
   return
end

while 1 % 1 equivale a true: bulce indefinido

    % En cada iteracion va llamando a los datos de cada linea una ...
a una en
    % la variable tline que es de tipo 'char'.
    tline = fgetl(fid);

    % Finalizamos el bucle cuando tline no lea nada. Entonces
devolvera el
```

```
    % valor -1 que al ser de tipo 'double' hara saltar break.
    if ~ischar(tline)
        break
    end

    % Transformamos cada linea a un cell y lo convertimos en valores
    % numericos para usar en una matriz. La funcion textscan ...
devolvera un
    % conjunto vacio  de celdascuando no coincida con el patron dado.
    % Entonces no se incuira el texto en matdata.
    cellinfo = textscan(tline,'%f, %f, %f, %f, %f');
    matinfo  = cell2mat(cellinfo);

    % match fails for text lines, textscan returns empty cells
    datos = [datos; matinfo];

end

% Cerramos el archivo de texto.
fclose(fid);

% Mostramos los valores
signals = unique(datos(:,1))';
set(handles.text6,'string',mat2str(signals));

% Determinamos posiciones
[n,~] = size(datos);
p     = length(signals);

pos_ini = [];
datos = [datos(:,1:3), sqrt(datos(:,4).^2 + datos(:,5).^2)];

for j = 1:p
   for k = 1:n
       if datos(k,1) == signals(j)
           pos_ini = [pos_ini; datos(k,:)];
           break
       end
   end
end

global pos;
pos = pos_ini;

for j = 1:p
   for k = 1:n
       if datos(k,1) == signals(j) && pos(j,4)>datos(k,4)
           pos(j,:) = datos(k,:);
       end
   end
   [X(j),Y(j),Z(j)] = proyeccion(pos(j,2),pos(j,3));
end

% Pintamos los puntos observados
hold on, plot3(X,Y,Z,'og','Linewidth',3);
```

```
for j = 1:p
    hold on,
text(X(j),Y(j),Z(j)+1,mat2str(signals(j)),'Color','red', ...
'FontSize',20)
end

function pushbutton2_Callback(hObject, eventdata, handles)

global pos; % Cada funcion que la use debe llamarla como global.

if isempty(pos)
    disp(' ');
    disp('¡ERROR, NO SE HA LEIDO EL ARCHIVO DE DATOS!')
    disp('Sugerencias: ');
    disp('- Realice la lectura del archivo de datos .txt.')
    return
end

recorrido = get(handles.edit2,'string');
periodo   = get(handles.edit3,'string');
alpha     = get(handles.edit4,'string');

rec_mat   = str2num(recorrido);
alpha_mat = str2num(alpha);
per_mat   = str2num(periodo);

if length(alpha_mat) >= length(rec_mat) || length(per_mat) >=
length(rec_mat)
    disp(' ');
    disp('¡ERROR AL INTRODUCIR LAS COMPONENTES DE \alpha y Periodo!')
    disp('Sugerencias: ');
    disp('- Revise el numero de componentes del alpha y Periodo');
    disp('- Recuerde que el numero de componentes debe ser el ...
numero de componentes de recorrido -1');
    return;
end

for j = 1:length(alpha_mat)
    if alpha_mat(j) == 1
        indicador(j) = 1;
    else
        indicador(j) = 0;
    end
end

dimension     = mat2str(length(indicador));
j_max         = mat2str(length(indicador)-1);
indicador     = regexprep( mat2str(indicador), {'\[', '\]', '\s+'}...
, {'', '', ','});
periodo       = regexprep( mat2str(per_mat), {'\[', '\]', '\s+'}...
, {'', '', ','});
alpha         = regexprep( mat2str(alpha_mat), {'\[', '\]', '\s+'}...
, {'', '', ','});

% Asignacion de ruta
```

```
[n, ~] = size(pos);

for j = 1:length(rec_mat)
    for k = 1:n
        if pos(k,1) == rec_mat(j)
            pos_rec(j,:) = pos(k,:);
        end
    end
end

ang_pan_ini   = regexprep( mat2str(pos_rec(1:(end-1),2)'), ...
 {'\[', '\]', '\s+'}, {'', '', ','});
ang_tilt_ini  = regexprep( mat2str(pos_rec(1:(end-1),3)'), ...
 {'\[', '\]', '\s+'}, {'', '', ','});
ang_pan_fin   = regexprep( mat2str(pos_rec(2:end,2)'), ...
 {'\[', '\]', '\s+'}, {'', '', ','});
ang_tilt_fin  = regexprep( mat2str(pos_rec(2:end,3)'), ...
 {'\[', '\]', '\s+'}, {'', '', ','});

recorrido_txt(indicador,alpha,periodo,ang_pan_ini,ang_tilt_ini, ...
ang_pan_fin,ang_tilt_fin,dimension,j_max);

%% Grafico %%

function axes1_CreateFcn(hObject, eventdata, handles)
x0 = 21.47;
y0 = 0;

d  = 38;
c  = 7.5;
e  = 6;
f  = 12.5;
alpha = atan((c - e)/f);

P_cam = [];
P_cam_v = [];
P_cam_v_inf = [];
P_cam_v_sup = [];

for ang = -45:1:45;

    beta = deg2rad(ang);
    phi = beta + alpha;

    % Punto entre visagra y placa
    x_1 = e*cos(phi) + x0;
    y_1 = -e*sin(phi) + y0;

    % Punto de la camara
    x_cam    = x_1 + d*cos(pi/2-phi);
    y_cam    = y_1 + d*sin(pi/2-phi);
    P_cam    = [P_cam; x_cam, y_cam];

    p = 4;
    % Punto para el vector de apuntado
```

```
    x_cam_v  = x_cam + p*cos(phi);
    y_cam_v  = y_cam - p*sin(phi);
    P_cam_v  = [P_cam_v; x_cam_v, y_cam_v];

    % Vector de apertura de apuntado superior
    x_cam_v_sup = x_cam + p*cos(phi - deg2rad(23.5));
    y_cam_v_sup = y_cam - p*sin(phi - deg2rad(23.5));
    P_cam_v_sup = [P_cam_v_sup; x_cam_v_sup, y_cam_v_sup];

    % Vector de apertura de apuntado inferior
    x_cam_v_inf = x_cam + p*cos(phi + deg2rad(23.5));
    y_cam_v_inf = y_cam - p*sin(phi + deg2rad(23.5));
    P_cam_v_inf = [P_cam_v_inf; x_cam_v_inf, y_cam_v_inf];

end

psi = linspace(25,155,100);

for j = 1:100
    X(:,j) = P_cam(:,1)*cos(deg2rad(psi(j)));
    Y(:,j) = P_cam(:,1)*sin(deg2rad(psi(j)));
    Z(:,j) = P_cam(:,2);
end

surface(X,Y,Z);
colormap(gris);
shading interp;
grid on;
view(225,25);

%% Funciones adicionales %%

function [X,Y,Z] = proyeccion(pos_pan,pos_tilt)

factor_tilt = 94.92/1000;
factor_pan  = 130/1000;

x0 = 32.2;
y0 = -11.8;
l1 = 9.8;
l2 = 31.7;

d     = 38;
c     = 7.5;
e     = 6;
f     = 12.5;
R     = sqrt(f^2+(c-e)^2);
alpha = atan((c - e)/f);
t0    = deg2rad([0 40]);

P_cam       = [];
P_cam_v     = [];
P_cam_v_inf = [];
P_cam_v_sup = [];
```

```
for ang = 132.51 - pos_tilt*factor_tilt;

    thetas = fsolve(@(x) angulos(x,l1,l2,R,deg2rad(ang),x0,y0), t0);
clc;
    beta   = thetas(1);
    phi    = beta + alpha;

    % Nuevos puntos de partida
    x0 = 21.47;
    y0 = 0;

    % Punto entre visagra y placa
    x_1 = e*cos(phi) + x0;
    y_1 = -e*sin(phi) + y0;

    % Punto de la camara
    x_cam    = x_1 + d*cos(pi/2-phi);
    y_cam    = y_1 + d*sin(pi/2-phi);
    P_cam    = [P_cam; x_cam, y_cam];

    p = 4;
    % Punto para el vector de apuntado
    x_cam_v  = x_cam + p*cos(phi);
    y_cam_v  = y_cam - p*sin(phi);
    P_cam_v  = [P_cam_v; x_cam_v, y_cam_v];

    % Vector de apertura de apuntado superior
    x_cam_v_sup = x_cam + p*cos(phi - deg2rad(23.5));
    y_cam_v_sup = y_cam - p*sin(phi - deg2rad(23.5));
    P_cam_v_sup = [P_cam_v_sup; x_cam_v_sup, y_cam_v_sup];

    % Vector de apertura de apuntado inferior
    x_cam_v_inf = x_cam + p*cos(phi + deg2rad(23.5));
    y_cam_v_inf = y_cam - p*sin(phi + deg2rad(23.5));
    P_cam_v_inf = [P_cam_v_inf; x_cam_v_inf, y_cam_v_inf];

end

psi = 25 + pos_pan*factor_pan;

X = P_cam(:,1)*cos(deg2rad(psi));
Y = P_cam(:,1)*sin(deg2rad(psi));
Z = P_cam(:,2);

function g = gris
g = linspace(0.1,0.9,63)';
g = [g g g];

function F = angulos(x,l1,l2,R,t1,x0,y0)
%
beta = x(1);
t2   = x(2);
%
F(1) = l1*cos(t1) + l2*cos(t2 - t1) - R*sin(beta) - x0;
F(2) = -l1*sin(t1) + l2*sin(t2 - t1) - R*cos(beta) - y0;
```

```
function
recorrido_txt(indicador,alpha,periodo,ang_pan_ini,ang_tilt_ini, ...
ang_pan_fin,ang_tilt_fin,dimension,j_max)
fid = fopen('trayectorias.ino','wt');

string1 = '// Incluimos las librerias necesarias.';
string2 = '';
string3 = '#include <SPI.h>';
string4 = '#include <Pixy.h>';
string5 = 'Pixy Pixy;';
string6 = '';
string7 = '// Definicion de las variables.';
string8 = '';
string9 = 'boolean cont = true;                    // Definimos la ...
variable booleana de control que finaliza las trayectorias y para...
 los servos.';
string10 = '';
string11 = 'int j;                                  // Definimos la
cuenta de trayectorias (ej: j = 2 implica que tenemos 3 trayetorias
(0,1,2)).';
string12 = 'double t;                               // Definiimos el
tiempo local de cada trayectoria.';
string13 = '';
string14 = ['int              j_max = ',j_max ,';           // Porque
la cuenta de j empieza en 0.'];
 string15 = ['double    indicador[',dimension,'] = {',indicador,'};
// Indicamos si queremos trayectoria de tipo 0 () o de tipo 1 ().'];
string16 = ['double        alpha[',dimension,'] = {',alpha,'};     //
Valor de alpha de la trayectoria. Solo se emplean los valores si el
indicador vale 0. Para indicador 1, usamos 1.'];
string17 = ['double            T[',dimension,'] = {',periodo,'};
// Duracion de la trayectoria. '];
string18 = ['double ang_pan_ini[',dimension,']  = {',ang_pan_ini,'};
// Posiciones horizontales iniciales.'];
string19 = ['double ang_tilt_ini[',dimension,'] = {',ang_tilt_ini,'};
// Posiciones verticales iniciales.'];
string20 = ['double ang_pan_fin[',dimension,']  = {',ang_pan_fin,'};
// Posiciones horizontales finales.'];
string21 = ['double ang_tilt_fin[',dimension,'] = {',ang_tilt_fin,'};
// Posiciones verticales finales.'];
string22 = '';
string23 = '// Bucle de inicializacion (solo se ejecuta una vez).';
string24 = '';
string25 = 'void setup()';
string26 = '{';
string27 = '  Serial.begin(9600); // Inicializacion del serial.';
string28 = '  j = 0;              // Comenzamos con la trayectoria
0.';
string29 = '  t = 0;              // Se inicializa el tiempo en 0
para la trayectoria 0.';
string30 = '  Pixy.init();        // Inicializacion de Pixy.';
string31 = '}';
string32 = '';
string33 = '// Bucle principal';
string34 = '';
```

```
string35 = 'void loop() ';
string36 = '{';
string37 = '  double pan_serv;   // Definimos la posicion del servo
horizontal para uso en el bucle principal.';
string38 = '  double tilt_serv;  // Definimos la posicion del servo
vertical para uso en el bucle principal.';
string39 = '';
string40 = '  if (t == 0 && j < j_max) // Si nos encontramos al
inicio de la PRIMERA trayectoria (t=0).';
string41 = '  { ';
string42 = '    Pixy.setServos(ang_pan_ini[j],ang_tilt_ini[j]); //
Nos movemos a los inicios de trayectorias.';
string43 = '    delay(3000);';
string44 = '  }';
string45 = '';
string46 = '  if (cont == true) // Si la variable booleana permite
las trayectorias.';
string47 = '  {';
string48 = '';
string49 = '    // MANIOBRA CON PARES DESIGUALES';
string50 = '';
string51 = '    if (indicador[j] == 0)';
string52 = '    {';
string53 = '      if (t/T[j] <= alpha[j]) // Primera parte de la
trayectoria.';
string54 = '      {';
string55 = '        pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] -
ang_pan_ini[j])*pow(t,2)/pow(T[j],2)/alpha[j];';
string56 = '        tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] -
ang_tilt_ini[j])*pow(t,2)/pow(T[j],2)/alpha[j];  ';
string57 = '      }';
string58 = '      if (t/T[j] > alpha[j])  // Segunda parte de la
trayectoria.';
string59 = '      {';
string60 = '        pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] -
ang_pan_ini[j])*( alpha[j] + 2*( t/T[j] - alpha[j] ) + pow(( t/T[j] -
alpha[j] ),2)/(alpha[j] - 1) );';
string61 = '        tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] -
ang_tilt_ini[j])*( alpha[j] + 2*( t/T[j] - alpha[j] ) + pow(( t/T[j]
- alpha[j] ),2)/(alpha[j] - 1) );';
string62 = '      }';
string63 = '    }';
string64 = '';
string65 = '    // MANIOBRA DE MiNIMA ENERGiA';
string66 = '';
string67 = '    if (indicador[j] == 1)';
string68 = '    {';
string69 = '      pan_serv  = ang_pan_ini[j] + (ang_pan_fin[j] -
ang_pan_ini[j])*(3*T[j] - 2*t)*pow(t,2)/pow(T[j],3);';
string70 = '      tilt_serv = ang_tilt_ini[j] + (ang_tilt_fin[j] -
ang_tilt_ini[j])*(3*T[j] - 2*t)*pow(t,2)/pow(T[j],3);';
string71 = '    }';
string72 = '';
string73 = '    // POSICIONADO DE LOS SERVOS';
string74 = '';
```

```
string75 = '    Pixy.setServos(pan_serv,tilt_serv); // Movemos alas
posiciones calculadas. ';
string76 = '    delay(10);                          // Generamos un
retraso de 10 ms.';
string77 = '    t = t + 0.01;                       // Aumentamos el
tiempo para calcular la posicion tras esos 10 ms (es bastante
aproximado dado el tiempo de procesamiento).';
string78 = '    //Serial.println(t);';
string79 = '';
string80 = '    if (t >= T[j]) // Cuando se supere el periodo de la
trayectoria se finaliza.';
string81 = '    {';
string82 = '      j++;         // Siguiente trayectoria.';
string83 = '      t = 0;       // Tiempo local puesta a 0.';
string84 = '      delay(5000); // Pausa de X ms entre trayectoria y
trayectoria.';
string85 = '    }  ';
string86 = '  }';
string87 = '';
string88 = '  if (j > j_max)   // Cuando se acabe la ultima
trayectoria se deja de mover los servos.';
string89 = '  {';
string90 = '    cont = false;';
string91 = '  }';
string92 = '}';

fprintf(fid, '%s\n%s', string1, string2);
fprintf(fid,'\n%s\n%s', string3, string4);
fprintf(fid,'\n%s\n%s', string5, string6);
fprintf(fid,'\n%s\n%s', string7, string8);
fprintf(fid,'\n%s\n%s', string9, string10);
fprintf(fid,'\n%s\n%s', string11, string12);
fprintf(fid,'\n%s\n%s', string13, string14);
fprintf(fid,'\n%s\n%s', string15, string16);
fprintf(fid,'\n%s\n%s', string17, string18);
fprintf(fid,'\n%s\n%s', string19, string20);
fprintf(fid,'\n%s\n%s', string21, string22);
fprintf(fid,'\n%s\n%s', string23, string24);
fprintf(fid,'\n%s\n%s', string25, string26);
fprintf(fid,'\n%s\n%s', string27, string28);
fprintf(fid,'\n%s\n%s', string29, string30);
fprintf(fid,'\n%s\n%s', string31, string32);
fprintf(fid,'\n%s\n%s', string33, string34);
fprintf(fid,'\n%s\n%s', string35, string36);
fprintf(fid,'\n%s\n%s', string37, string38);
fprintf(fid,'\n%s\n%s', string39, string40);
fprintf(fid,'\n%s\n%s', string41, string42);
fprintf(fid,'\n%s\n%s', string43, string44);
fprintf(fid,'\n%s\n%s', string45, string46);
fprintf(fid,'\n%s\n%s', string47, string48);
fprintf(fid,'\n%s\n%s', string49, string50);
fprintf(fid,'\n%s\n%s', string51, string52);
fprintf(fid,'\n%s\n%s', string53, string54);
fprintf(fid,'\n%s\n%s', string55, string56);
fprintf(fid,'\n%s\n%s', string57, string58);
fprintf(fid,'\n%s\n%s', string59, string60);
```

```
fprintf(fid,'\n%s\n%s', string61, string62);
fprintf(fid,'\n%s\n%s', string63, string64);
fprintf(fid,'\n%s\n%s', string65, string66);
fprintf(fid,'\n%s\n%s', string67, string68);
fprintf(fid,'\n%s\n%s', string69, string70);
fprintf(fid,'\n%s\n%s', string71, string72);
fprintf(fid,'\n%s\n%s', string73, string74);
fprintf(fid,'\n%s\n%s', string75, string76);
fprintf(fid,'\n%s\n%s', string77, string78);
fprintf(fid,'\n%s\n%s', string79, string80);
fprintf(fid,'\n%s\n%s', string81, string82);
fprintf(fid,'\n%s\n%s', string83, string84);
fprintf(fid,'\n%s\n%s', string85, string86);
fprintf(fid,'\n%s\n%s', string87, string88);
fprintf(fid,'\n%s\n%s', string89, string90);
fprintf(fid,'\n%s\n%s', string91, string92);

fclose(fid);
```

Script 2: *Interfaz_vision.m*

Este script, junto con *Interfaz_vision.fig*, genera la interfaz gráfica de visión.

```
function varargout = Interfaz_vision(varargin)

clearvars -except varargin;

gui_Singleton = 1;
gui_State = struct('gui_Name',       mfilename, ...
                   'gui_Singleton',  gui_Singleton, ...
                   'gui_OpeningFcn', @Interfaz_vision_OpeningFcn, ...
                   'gui_OutputFcn',  @Interfaz_vision_OutputFcn, ...
                   'gui_LayoutFcn',  [] , ...
                   'gui_Callback',   []);
if nargin && ischar(varargin{1})
    gui_State.gui_Callback = str2func(varargin{1});
end

if nargout
    [varargout{1:nargout}] = gui_mainfcn(gui_State, varargin{:});
else
    gui_mainfcn(gui_State, varargin{:});
end

function Interfaz_vision_OpeningFcn(hObject, eventdata, handles,
varargin)

handles.output = hObject;

guidata(hObject, handles);
```

```
function varargout = Interfaz_vision_OutputFcn(hObject, eventdata,
handles)

varargout{1} = handles.output;

%% Editables

function edit1_Callback(hObject, eventdata, handles)

function edit1_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit2_Callback(hObject, eventdata, handles)

function edit2_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit3_Callback(hObject, eventdata, handles)

function edit3_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit4_Callback(hObject, eventdata, handles)

function edit4_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

function edit5_Callback(hObject, eventdata, handles)

function edit5_CreateFcn(hObject, eventdata, handles)
if ispc && isequal(get(hObject,'BackgroundColor'),
get(0,'defaultUicontrolBackgroundColor'))
    set(hObject,'BackgroundColor','white');
end

%% Pulsadores

function pushbutton1_Callback(hObject, eventdata, handles)
global IM_or

im = get(handles.edit1,'string');
IM_or = imread(im);
```

```
hold off, imshow(IM_or)

function pushbutton2_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = graythresh(IM_or);
IM_bin  = 1 - im2bw(IM_or,umbral);

hold off, imshow(IM_bin)

function pushbutton3_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = str2num(get(handles.edit2,'string'));

if isempty(umbral)
   disp('ERROR: No se ha introducido ningún valor del Umbral')
    return
end

IM_bin_cor  = 1 - im2bw(IM_or,umbral);

hold off, imshow(IM_bin_cor)

function pushbutton4_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = str2num(get(handles.edit2,'string'));

if isempty(umbral)
    disp('ERROR: No se ha introducido ningún valor del Umbral')
    return
end

IM_bin_cor  = 1 - im2bw(IM_or,umbral);
p = str2num( get(handles.edit3,'string') );

if isempty(p)
    disp('ERROR: No se ha introducido ningún valor del Tamaño
límite')
```

```
    return
end

IM_bin_ru  = bwareaopen(IM_bin_cor, p);

hold off, imshow(IM_bin_ru)

function pushbutton5_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = str2num(get(handles.edit2,'string'));

if isempty(umbral)
    disp('ERROR: No se ha introducido ningún valor del Umbral')
    return
end

IM_bin_cor  = 1 - im2bw(IM_or,umbral);
p = str2num( get(handles.edit3,'string') );

if isempty(p)
    disp('ERROR: No se ha introducido ningún valor del Tamaño
límite')
    return
end

IM_bin_ru  = bwareaopen(IM_bin_cor, p);
IM_bin_hu = imfill(IM_bin_ru,'holes');

hold off, imshow(IM_bin_hu)

function pushbutton6_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = str2num(get(handles.edit2,'string'));

if isempty(umbral)
    disp('ERROR: No se ha introducido ningún valor del Umbral')
    return
end

IM_bin_cor  = 1 - im2bw(IM_or,umbral);
p = str2num( get(handles.edit3,'string') );

if isempty(p)
```

```
    disp('ERROR: No se ha introducido ningún valor del Tamaño ...
límite')
    return
end

IM_bin_ru  = bwareaopen(IM_bin_cor, p);
IM_bin_hu = imfill(IM_bin_ru,'holes');

dimension = str2num( get(handles.edit4,'string') );
forma     = get(handles.edit5,'string');

se = strel(forma,dimension);
IM_bin_bo = imclose(IM_bin_hu,se);
hold off, imshow(IM_bin_bo)

function pushbutton7_Callback(hObject, eventdata, handles)
global IM_or

if isempty(IM_or)
   disp('ERROR: No se ha cargado la imagen');
   return
end

umbral = str2num(get(handles.edit2,'string'));

if isempty(umbral)
    disp('ERROR: No se ha introducido ningún valor del Umbral')
    return
end

IM_bin_cor  = 1 - im2bw(IM_or,umbral);
p = str2num( get(handles.edit3,'string') );

if isempty(p)
    disp('ERROR: No se ha introducido ningún valor del Tamaño
límite')
    return
end

IM_bin_ru  = bwareaopen(IM_bin_cor, p);
IM_bin_hu = imfill(IM_bin_ru,'holes');

dimension = str2num( get(handles.edit4,'string') );
forma     = get(handles.edit5,'string');

se = strel(forma,dimension);
IM_bin_bo = imclose(IM_bin_hu,se);

% - Bordes
[B,L] = bwboundaries(IM_bin_bo,'noholes');
hold off,imshow(label2rgb(L, @jet, [.5 .5 .5]))
hold on
for k = 1:length(B)
    boundary = B{k};
    plot(boundary(:,2), boundary(:,1), 'w', 'LineWidth', 2)
```

```
end

% - Centroide
stats = regionprops(L,'Area','Centroid');

for k = 1:length(B)
    centroid = stats(k).Centroid;
    hold on, plot(centroid(1),centroid(2),'ko');
end
```

REFERENCIAS

[1] R. Vázquez Valenzuela, Apuntes de Dinámica de Vehículos Espaciales, Sevilla: Sección de Publicaciones, Escuela Técnica Superior de Ingniería, Universidad de Sevilla, 2017.

[2] «PIXY,» [En línea]. Available: https://Pixycam.com/.

[3] «PIXY Documentation,» [En línea]. Available: https://docs.Pixycam.com/.

[4] GOTECK RC INC, «http://www.goteckrc.com/Download/Micro%20Servo/GS-9018%20revised.pdf,» 27 02 2013. [En línea].

[5] «PIXY Documentation,» [En línea]. Available:
https://docs.Pixycam.com/wiki/doku.php?id=wiki:v1:building_the_pan_tilt_1.

[6] «CMUcam5 Pixy,» [En línea]. Available: http://www.cmucam.org/projects/cmucam5/wiki/Latest_release.

[7] «ARDUINO,» [En línea]. Available: https://www.arduino.cc/en/Main/Software.

[8] A. O. Baturone, Robótica: manipuladores y robots móviles.

[9] «CMUcam5 Pixy,» [En línea]. Available: http://www.cmucam.org/projects/cmucam5.