UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

Documentos:

1-Memoria

2-Pliego de condiciones

3-Presupuesto

Apéndice 1-Documentación del código de la aplicación

Apéndice 2-Código

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

Resumen

El uso de robots es cada vez mayor en la industria y los hay de distintos tipos, en concreto en este proyecto se presenta un robot delta, que es un tipo de robot para pick and place de muy alta precisión y velocidad, pero zona de trabajo reducida, lo que los convierte en robots ideales para trabajar en lineas de empaquetado automatizadas. Algunos modelos de este tipo de robots son el flexy picker de ABB o el YF03 de Kawasaki.

Este proyecto pretende recrear uno de estos robots adicionalmente con un sistema de visión artificial de bajo coste y hecho en la medida de los posible con software y hardware libre.

Para este proyecto especificamente se ha utilizado Arduino UNO para control de los servos e interactuar con la cámara, Qt creator para crear una aplicación con interfaz de usuario y cálculo de la cinemática y Pixy camera, una cámara basada en ARM cortex-M especificamente diseñada para aplicaciones de visión artificial.

Palabras clave: robot delta, Arduino, Pixy camera, hardware libre, software libre, visión artificial, Qt, ARM Cortex-M

Resum

L'ús de robots és cada vegada major en la indústria i hi ha diferent tipus, en concret en aquest projecte es presenta un robot delta, que és un tipus de robot per a pick and place amb una precisió i velocitat molt alta, però amb una zona de treball reduïda, el que els converteix en robots ideals per a treballar en línies d'empaquetament automatitzades. Alguns models d'aquest tipus de robots són el "flexy picker" d'ABB o el "YF03" de Kawasaki.

Aquest projecte pretén recrear un d'aquests robots, addicionalment amb un sistema de visió artificial de baix cost i fet en la mesura del possible amb software i hardware lliure.

Per a aquest projecte s'ha utilitzat Arduino Uno, per al control del servos i interactuar amb la càmera, Qt creator per a crear una aplicació amb interfície d'usuari i el càlcul de la cinemàtica i Pixy camera, una càmera basada en ARM cortex-M específicament dissenyada per a aplicacions de visió artificial.

Paraules clau: robot delta, Arduino, Pixy camera, hardware lliure, software lliure, visió artificial, Qt, ARM Cortex-M

Abstract

The use of robots is growing day by day in the industry and there are different kinds of them, in this project a delta robot is presented, delta robots are a kind of pick and place robots with a very high accuracy and velocity, but with a small working zone, what makes them ideal for working in automated packaging lines. Some of the models of this kind of robots are "flexy picker" from ABB or "YF03" from kawasaki.

This project aims to recreate one of those robots, additionally with a low cost computer vision system and made as much as possible with open source hardware and software.

For this project has been specifically used Arduino Uno for controlling the servos and electromagnet, and interfacing with the camera, Qt creator in order to create an application with a user interface and calculate the kinematics, and Pixy camera, an ARM cortex-M based camera, specifically designed to be used on computer vision application.

Keywords: delta robot, Arduino, Pixy camera, open source hardware, opensource software, computer vision, Qt, ARM Cortex-M

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

1. Memoria

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

Índice

Lista de Figuras

Lista de tablas

1 Introducción a los robot delta

1.1 ¿Qué es un robot delta?

Un robot delta es un tipo de robot paralelo que consiste en tres brazos conectados por articulaciones universales a la base. La clave del diseño es el uso de paralelogramos en los brazos, que mantienen la orientación del efector final. Delta robots wikipedia [1]

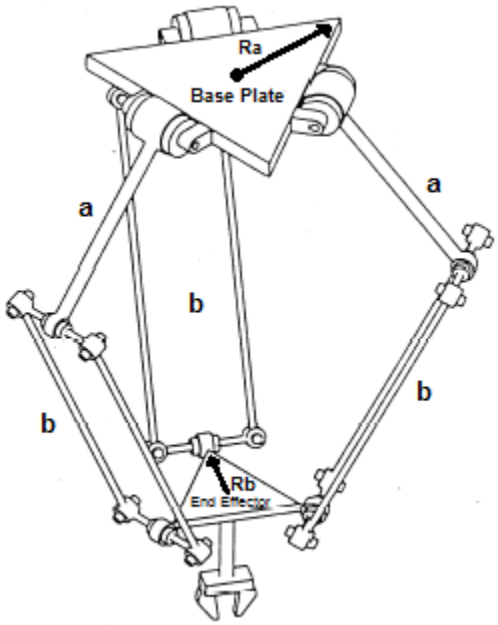


Figura 1: Robot delta

1.2 Breve historia de los robots delta

El robot delta fue inventado a principios de los noventa por equipo de investigación liderado por el profesor Reymond Clavel en la École Polytechnique Fédérale de Lausanne (EPFL, Suiza). El propósito de este robot no era otro que el de manipular objetos pequeños y ligeros a una velocidad muy alta. En 1987 la compañía suiza Demareux compró una licencia del robot delta y comenzó la producción de robots delta para la industria de empaquetamiento. En 1991 Reymond Clavel presentó su tesis doctoral 'Conception d'un robot parallèle rapide à 4 degrés de liberté' y recibió el premio del golden robot en 1999 por su trabajo y desarrollo del robot delta.

Más tarde otras compañías comprarían la patente a la universidad y también comenzarían a producir robots delta.

2 Factores a considerar

2.1 Especificaciones

- Robot de pick and place basado en la medida de lo posible en software y hardware libre
- Con visión artificial
- Interfaz gráfica fácil e intuitiva

- Diferentes modos de funcionamiento, manual y automático
- El sistema ha de tener la capacidad de comandar hardware tipo motores... por lo que es necesario un micro controlador
- El sistema ha de ser capaz de realizar cálculos de cinemática en el mínimo tiempo posible, por lo que es necesario un ordenador
- El sistema ha de ser multiplataforma

2.2 Alternativas

2.2.1 Alternativas al ordenador

Hoy en día existen multitud de ordenadores de bajo coste y consumo energético basados en micro-controladores ARM y compatibles con Qt, a continuación se propone una pequeña lista de estos y sus especificaciones.

	Raspberry Pi B	A10-OLinuXino	BeagleBone Black	ODROID-U3	Cubieboard 2
Processor	ARM11 700MHz	Cortex-A8 1GHz	Cortex-A8 1GHz	Cortex-A9 Quad 1.7GHz	Cortex-A7 Dual 1GHz
GPU	Video Core IV	Mali-400	SGX530	Mali-400	Mali-400
RAM	512MB	512MB	512MB	2GB	1GB
Flash	0MB	0MB	2GB	0MB	4GB
USB	2	2 host + 1 OTG	1 host + 1 OTG	3 host + 1 OTG	2 host + 1 OTG
Ethernet	1×10/100M	1×10/100M	1×10/100M	1×10/100M	1×10/100M
WiFi	✗	✗	✗	✗	✗
Bluetooth	✗	✗	✗	✗	✗
SATA	✗	✓	✗	✗	✓
GPIO	17	160	65	Expansion	96
SPI	✓	✓	✓	Expansion	✓
I2C	✓	✓	✓	Expansion	✓
Price	$35 USD	30€ (≈ $41 USD)	$45 USD	$59 USD	$65 USD
Comment	Large Community		Large Community		

Tabla 1: Alternativas al ordenador

2.2.2 Alternativas para el microcontrolador

Hoy en día el mercado ofrece multitud de placas de desarrollo para microcontroladores con procesadores de más o menos bits y distintas frecuencias de reloj, a continuación se propone una pequeña

lista con algunos ejemplos y sus especificaciones

	Arduino Mega 2560	STM32F407	LPC812	PIC32-PINGUINO-MICRO	Piccolo Launchpad
Manufacturer	Arduino	St microelectronics	NXP Semiconductors	Microchip and Olimex	Texas Instruments
Processor	ATMega 2560	ARM Cortex M4	ARM Cortex M0+	PIC32MX440F256H	ARM Cortex M4
Processor bits	8	32	32	32	32
Clock frequency	16MHz	168MHz	12MHz	80MHz	60MHz
Flash	256KB	up to 1MB	16 KB	256KB	64KB
SPI	Yes	Yes	Yes	Yes	Yes
I2C	Yes	Yes	Yes	Yes	Yes
GPIO	70	98	54	40	22
Price	$37.03 USD	14.47€ (≈ $16.38 USD)	$18.75 USD	12.95€ (≈ $14.66 USD)	$17.00(USD)
Comment	Large community and easiness	Large community			Large community

Tabla 2: Alternativas para el microcontrolador

2.2.3 Alternativas para la visión artificial

Existen múltiples alternativas para la resolver problemas de visión artificial, pero muy pocas de código libre, que es uno de los requisitos del proyecto.

La única alternativa opensoruce es OpenCV [2] que son unas bibliotecas publicadas bajo licencia BSD, para usar con C++, C, Python y Java, y con soporte para Windows, Linux, Mac OS, iOS y Android. OpenCV fue diseñado para eficiencia computacional y fuertemente centrado en aplicaciones de tiempo real. Escrito en C/C++, la biblioteca pude aprovechar el procesamiento multi núcleo. Usado en todo el mundo cuenta con una comunidad de más de cuarenta y siete mil usuarios. Sus usos van desde el arte interactivo, a inspección de minas, pasando por la robótica.

Figura 2: Logo de OpenCV

2.3 Solución adoptada

- Robot paralelo de pick and place tipo delta, no hay ningún motivo en especial para esta decisión, simplemente nos pareció interesante y hay disponibilidad de modelos para impresión en 3D.
- Visión artificial con cámara "Pixy CMU5 camera" [3], aunque es necesario la adquisición de un hardware especifico, esta cámara proporciona todas las bibliotecas necesarias para su funcionamiento y es extremadamente fácil de utilizar, es un proyecto de código abierto y multiplataforma.
- Interfaz gráfica hecha con Qt [4], es un proyecto muy potente para la creación de aplicaciones de C/C++, ya sabía manejarlo, es de códogo abierto y multiplataforma.
- Actuadores comandados con Arduino [5] Uno, extremadamente fácil de usar, compatible con Pixy CMU5 camera y con bibliotecas para gestionarla, de código abierto y entorno de desarrollo multiplataforma.
- La mayor carga de computación es realizada con un ordenador comunicado en serie con el Arduino Uno, se escogió un PC portátil estándar por disponibilidad y comodidad.

3 Hardware

3.1 Robot

Es la base de trabajo para el proyecto.

3.1.1 Caracterísiticas de los robot delta

Los robot delta son robots paralelos. Un robot paralelo está constituido por un efector final con n grados de libertad y una base fija, unidos por al menos dos cadenas cinemáticas independientes. El movimiento es ejecutado por n actuadores simples.

Este tipo de mecanismos es interesante porque cuando los actuadores están bloqueados, el efector final permanece es su sitio (esto es una medida importante de seguridad), y porque el número de actuadores es mínimo.

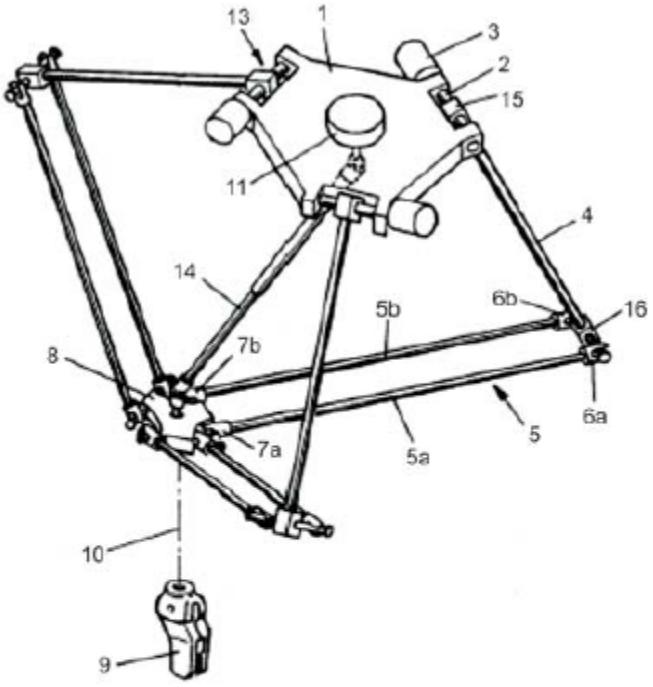
La idea básica tras el diseño del robot delta es el uso de paralelogramos. Un paralelogramo hace que una barra de salida permanezca en la misma posición con respecto a su barra de entrada. El uso de tres paralelogramos restringe completamente la orientación a únicamente tres grados de libertad unicamente de traslación. Las barras de entrada están montadas en levas rotatorias a través de articulaciones de revolución las uniones de revolución pueden ser movidas por servomotores CA o CC o con actuadores lineales. Finalmente, se una un mecanismo para transmitir el movimiento de rotación al efector final montado en la plataforma móvil.

El uso de actuadores montados en la base y barras de poca masa permiten conseguir a la plataforma móvil aceleraciones de hasta 50g en entornos experimentales y de 12g a 15g en aplicaciones industriales.

Los robot delta están especialmente indicados para aplicaciones de pick and place de alta velocidad y objetos de pequeña masa y geometrías simples.

Sin embargo su alta aceleración no los hace aptos para según que aplicaciones, pues los objetos a mover podría resultar dañados.

Para el agarre de objetos se suelen utilizar bombas de vacío por su bajo peso y su rápida actuación; aunque en este proyecto se va a utilizar un electroimán por ser mas sencillo de comandar y menos aparatoso de montar. Delta robots - Robots for high speed manipulation [6].



1 - Base element
2 - Shaft
3 - Fixed parts
4 - Arm
5a, 5b - Linking bars
6a, 6b - Revolute joints
7a, 7b - Revolute joints
8 - Movable element
9 - Working element
10 - End-effector joint
11 - Fixed motor
12 - Control system
13 - Actuator
14 - Telescopic arm (optional)
15 - First extremity
16 - Second extremity

Figura 3: Partes del robot delta (extraido de la paternte de EEUU 4.976582)

3.2 Cinemática de los robots delta

La cinemática se divide en dos problemas distintos. Cuando se sabe donde se quiere llevar el efector final y hay que saber en que posición deberán estar los motores para alcanzar dicha posición (cinemática inversa). Cuando se sabe la posición en la que están los actuadores y se quiere saber en que lugar estará el efector final (cinemática directa). Si se mira la figura 4 se puede ver que las plataformas son dos triángulmóvilos equiláteros, el triangulo verde es la plataforma fija, que contiene los actuadores y el triangulo rosa es la plataforma móvil, que contiene el efector final. El ángulo de las articulaciones de los actuadores es $theta_1$, $theta_2$ y $theta_3$, y el punto E_0 es la posición del efector final con las coordenadas (x_0, y_0, z_0).Delta robots - Robots for high speed manipulation.

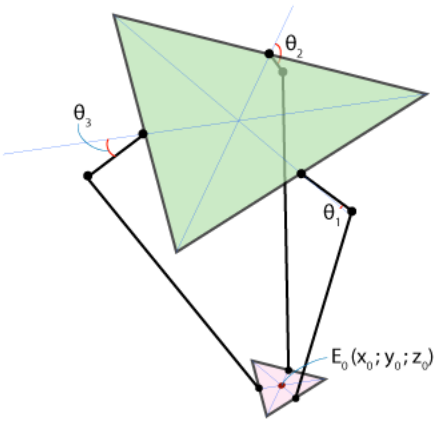


Figura 4: Parámetros de movimientos de las articulaciones de los robots delta

3.2.1 Cinemática inversa

Para solucionar la cinemática inversa es necesario crear una función que devuelva $theta_1$, $theta_2$ y $theta_3$ a partir de $E_0(x_0, y_0, z_0)$.

Primero hay que determinar algunos parámetros clave de la geometría del robot. Se tomará el lado del triangulo de la plataforma fija como ***f***, el lado del triangulo de plataforma móvil como ***e***, la longitud de la barra que está unida a la plataforma fija como ***rf*** y la longitud de la barra que está unida a la plataforma móvil como como ***re***. Todos los parámetros pueden ser observados en la figura 5. Estos parámetros dependerán del diseño del robot. Se tomará como origen de coordenadas el centro del triangulo de la plataforma fija, por lo que la coordenada ***z*** siempre será negativa.

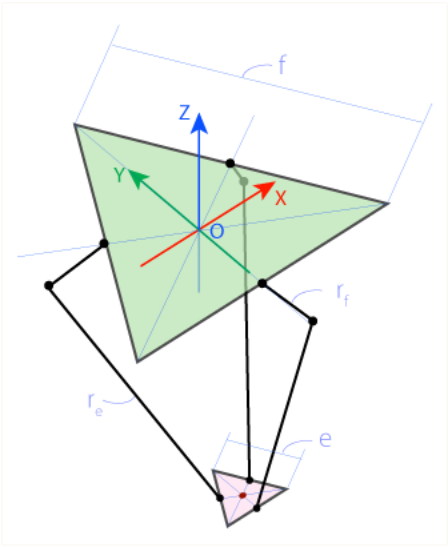


Figura 5: Parmetros de la geometría del robot

Debido al diseño del robot, la barra ***F1J1*** (ver la siguiente figura) solo puede rotar en el plano ***YZ*** formando un circulo con centro en ***F1*** y radio ***rf*** . Al contrario que ***F1***, ***J1*** y ***E1*** son articulaciones universales, lo que significa que ***E1J1*** puede rotar libremente relativo a ***E1***, formando una esfera con centro en ***E1*** y radio ***re***.

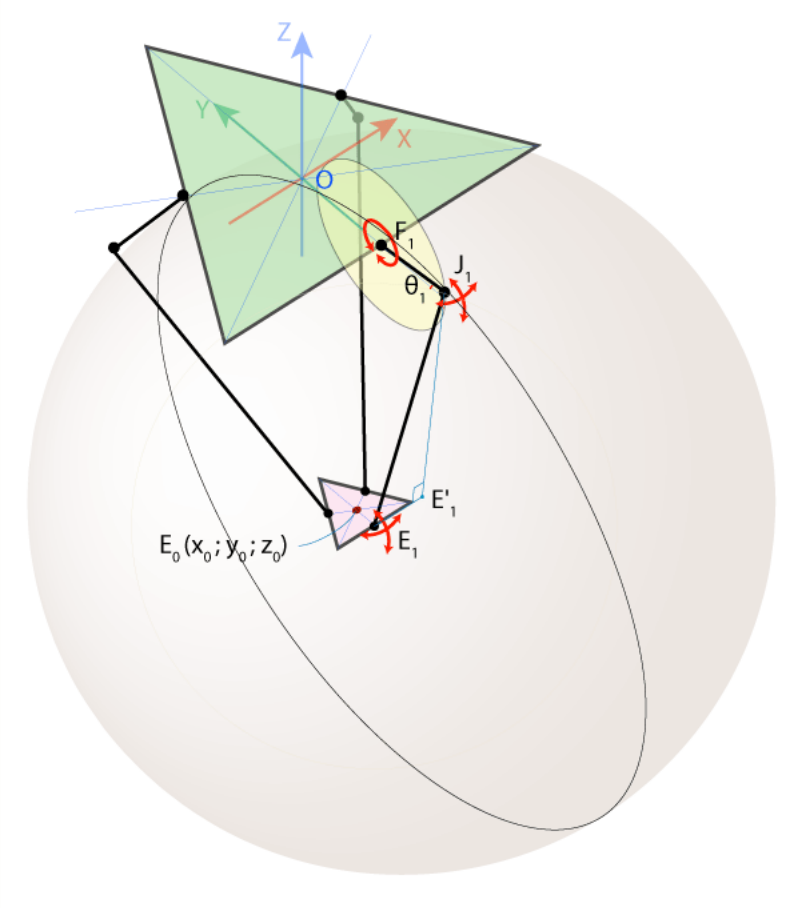


Figura 6

La intersección entre esta esfera y el plano ***YZ*** es un círculo con centro en el punto ***E'1*** y radio ***E'1J1***, dond ***E'1*** es la proyección del punto ***E1*** en el plano ***YZ***. El punto ***J1*** ahora puede ser encontrado como la intersección de dos círculos de radio conocido con centros en ***E'1*** y ***F1***. Y si se conoce ***J1***, se puede calcular el ángulo $\boldsymbol{theta_1}$.

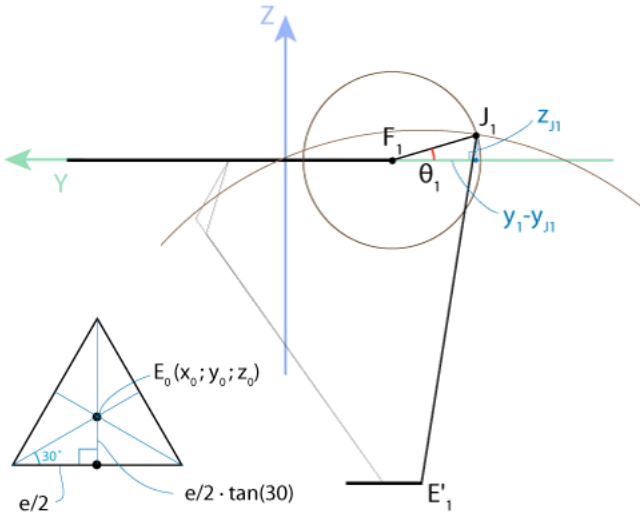


Figura 7: Plano YZ

$$E(x_0, y_0, z_0)$$

$$EE_1 = \frac{e}{2}\tan 30° = \frac{e}{2\sqrt{3}}$$

$$E_1(x_0, y_0 - \frac{e}{2\sqrt{3}}, z_0) \Rightarrow E'_1(0, y_0 - \frac{e}{2\sqrt{3}}, z_0)$$

$$E_1E'_1 = x_0 \Rightarrow E'_1J_1 = \sqrt{E_1J_1{}^2 - E_1E'_1{}^2} = \sqrt{r_e^2 - x_0^2}$$

$$F_1(0, -\frac{f}{2\sqrt{3}}, 0)$$

$$\begin{cases} (y_{J1} - y_{F1})^2 + (z_{J1} - z_{F1})^2 = r_f^2 \\ (y_{J1} - y_{E1})^2 + (z_{J1} - z_{E1})^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow \begin{cases} (y_{J1} + \frac{f}{2\sqrt{3}})^2 + z_{J1}{}^2 = r_f^2 \\ (y_{J1} - y_0 + \frac{e}{2\sqrt{3}})^2 + (z_{J1} - z_0)^2 = r_e^2 - x_0^2 \end{cases} \Rightarrow J_1(0, y_{J1}, z_{J1})$$

$$\theta_1 = \arctan\left(\frac{z_{J1}}{y_{F1} - y_{J1}}\right)$$

Figura 8: Obtencion de theta1

La barra ***F1J1*** solo se mueve en el plano ***YZ*** por lo que la coordenada ***X*** puede ser omitida. Para aprovechar esto en el resto de ángulos, ***theta$_2$*** y ***theta$_3$***, se debería utilizar la simetría del robot delta. Primero habrá que rotar el sistema de coordenadas en ***XY*** 120 grados en sentido anti horario, como se muestra en la siguiente figura.

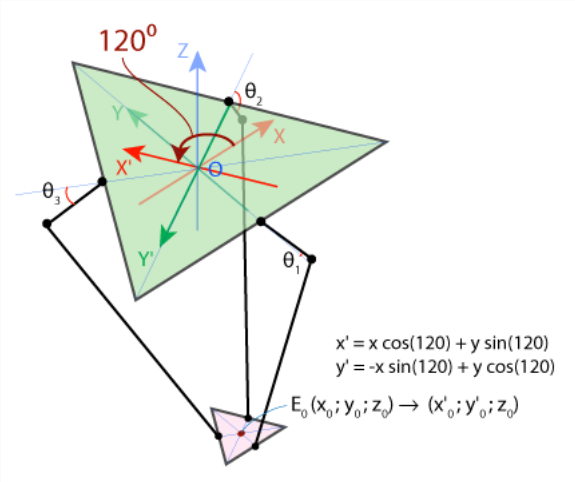


Figura 9: Rotando 120 grados

Se tiene un nuevo sistema de referencia ***X'Y'Z'***, y en este nuevo sistema de coordenadas podemos encontrar el ángulo ***theta₂*** usando el mismo algoritmo que se usó en para sacar ***theta₁***. El único cambio está en la necesidad de determinar ***x'0*** e ***y'0*** y esto se puede hacer utilizando la correspondiente matriz de rotación. Para encontrar el ángulo ***theta₃*** hay que rotar el sistema de referencia incial 120 grados en sentido horario. Delta robot kinematics [7].

3.2.2 Cinemática directa

Para solucionar la cinemática directa es necesario crear una función que devuelva $E_0(x_0, y_0, z_0)$ a partir de $theta_1$, $theta_2$ y $theta_3$.

Como los ángulos theta son conocidos, se pueden encontrar fácilmente las coordenadas de los puntos ***J1***, ***J2*** y ***J3*** (ver la figura mas abajo). Las barras ***J1E1***, ***J2E2*** y ***J3E3*** pueden rotar libremente alrededor de los puntos ***J1***, ***J2*** y ***J3*** respectivamente, formando tres esferas con radio ***re***.

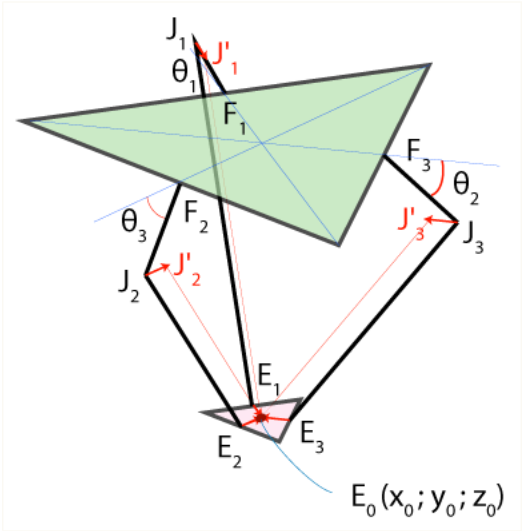


Figura 10

Si se desplazan los centros de las esferas de los puntos ***J1***, ***J2*** y ***J3*** a los puntos ***J'1***, ***J'2*** y ***J'3*** utilizando los vectores de transición ***E1E0***, ***E2E0*** y ***E3E0*** respectivamente, después de esta transición las tres esferas se cortarán en el mismo punto, ***E0***, como se muestra en la figura de abajo.

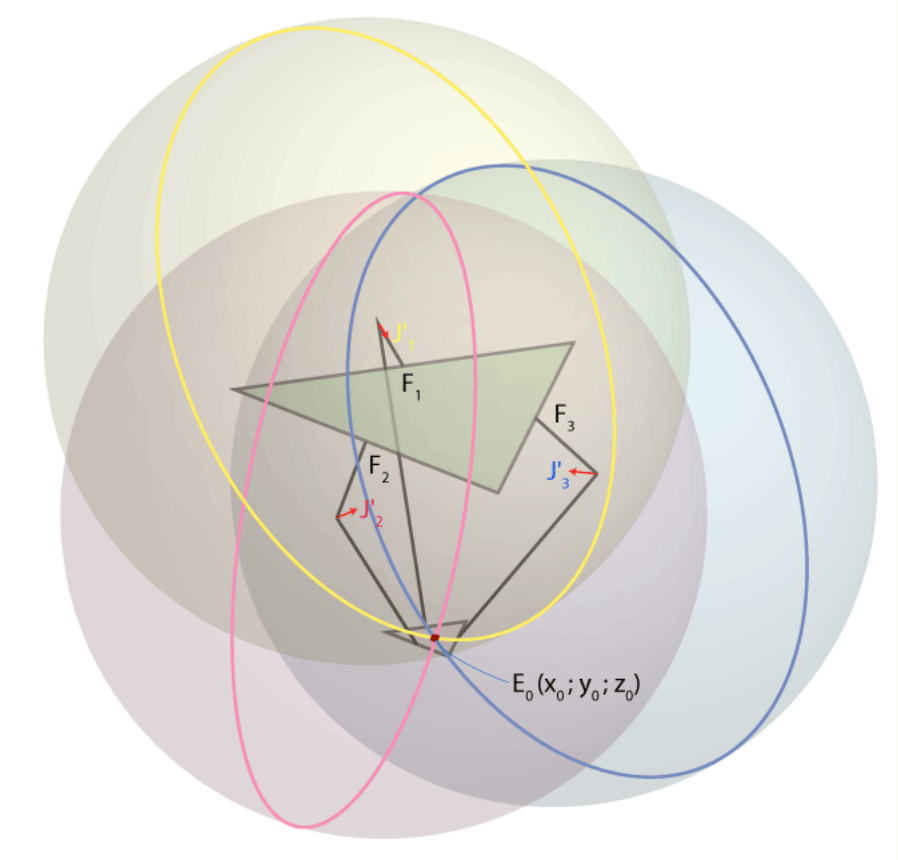


Figura 11

Así que, para encontrar las coordenadas ***(x0, y0, z0)***. del punto ***E0***hay que resolver tres ecuaciones tipo $(x - xj)^2 + (y - yj)^2 + (z - zj)^2 = re^2$ donde las coordenadas de los centros de las esferas, ***(xj, yj, zj)*** y los radios ***re*** son conocidos.

Primero hay que encontrar las coordenadas de los puntos ***J'1, j'2 y J'3***:

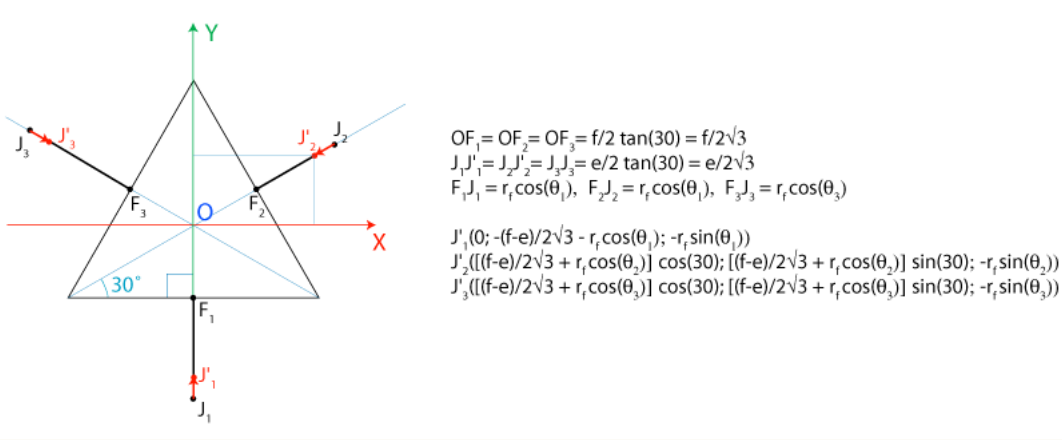


Figura 12

En las siguientes ecuaciones se designan las coordenadas de los puntos ***J1***, ***J2*** y ***J3*** como ***(x1, y1, z1)***, ***(x2, y2, z2)*** y ***(x3, y3, z3)***. Nótese que ***x0=0***. Estas son las ecuaciones de las tres esferas:

$$\begin{cases} x^2+(y-y_1)^2+(z-z_1)^2=r_e^2 \\ (x-x_2)^2+(y-y_2)^2+(z-z_2)^2=r_e^2 \\ (x-x_3)^2+(y-y_3)^2+(z-z_3)^2=r_e^2 \end{cases} \Rightarrow \begin{cases} x^2+y^2+z^2-2y_1y-2z_1z=r_e^2-y_1^2-z_1^2 & (1) \\ x^2+y^2+z^2-2x_2x-2y_2y-2z_2z=r_e^2-x_2^2-y_2^2-z_2^2 & (2) \\ x^2+y^2+z^2-2x_3x-2y_3y-2z_3z=r_e^2-x_3^2-y_3^2-z_3^2 & (3) \end{cases}$$

$$w_i = x_i^2+y_i^2+z_i^2$$

$$\begin{cases} x_2x+(y_1-y_2)y+(z_1-z_2)z=(w_1-w_2)/2 & (4)=(1)-(2) \\ x_3x+(y_1-y_3)y+(z_1-z_3)z=(w_1-w_3)/2 & (5)=(1)-(3) \\ (x_2-x_3)x+(y_2-y_3)y+(z_2-z_3)z=(w_2-w_3)/2 & (6)=(2)-(3) \end{cases}$$

From (4)-(5):

$$x=a_1z+b_1 \quad (7) \qquad y=a_2z+b_2 \quad (8)$$

$$a_1=\frac{1}{d}[(z_2-z_1)(y_3-y_1)-(z_3-z_1)(y_2-y_1)] \qquad a_2=-\frac{1}{d}[(z_2-z_1)x_3-(z_3-z_1)x_2]$$

$$b_1=-\frac{1}{2d}[(w_2-w_1)(y_3-y_1)-(w_3-w_1)(y_2-y_1)] \qquad b_2=\frac{1}{2d}[(w_2-w_1)x_3-(w_3-w_1)x_2]$$

$$d=(y_2-y_1)x_3-(y_3-y_1)x_2$$

Now we can substitute (7) and (8) in (1):

$$(a_1^2+a_2^2+1)z^2+2(a_1+a_2(b_2-y1)-z_1)z+(b_1^2+(b_2-y_1)^2+z_1^2-r_e^2)=0$$

Figura 13

3.2.3 Diseño del robot delta del proyecto

Como diseño del robot se ha tomado un modelo publicado en la página thingiverse [8] para impresión en 3D, "Delta Robot v2 with three arms"[9], y compuesto de las siguientes piezas:

- 12x articulaciones universales

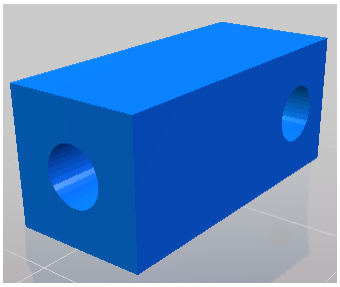


Figura 14: Articulación universal

- 3x bíceps como unión entre los actuadores y las uniones universales

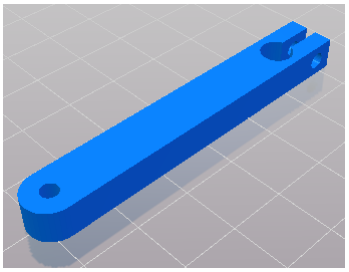


Figura 15: Bíceps

- 6x codos para la unión entre las articulaciones universales y los bíceps

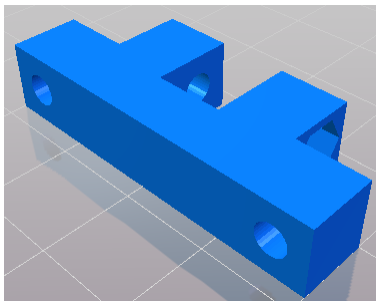


Figura 16: Codo

- 3x trozos de base, piezas que encajan entre entre ellas para formar una base triangular con ranuras para poner los actuadores

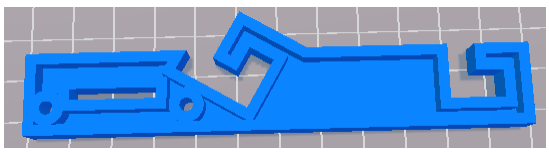


Figura 17: Trozo de base

- 1x efector final

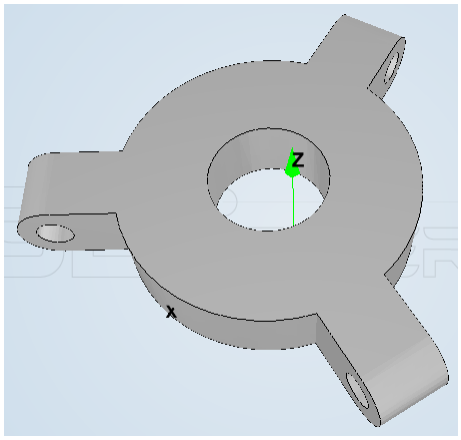


Figura 18: Efector final

- Tuercas, tornillos y varilla roscada
- 3x Servos TowerPro MG995 como actuadores

Figura 19: Servo

- 1x electroiman solenoidal

Figura 20: Electroimán

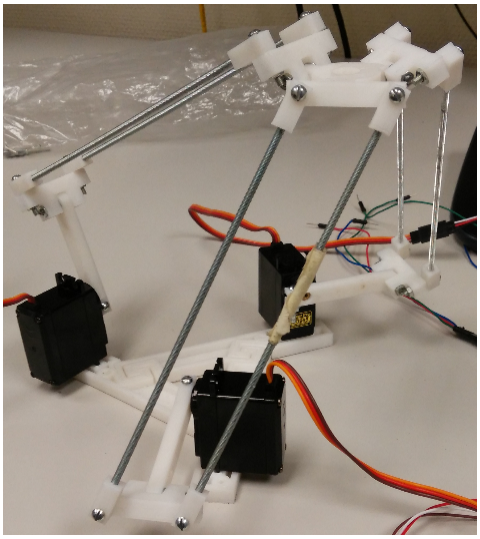


Figura 21: Robot delta montado

Los parámetros geométricos del robot son: e = 64mm, f = 220mm, re = 175mm, rf = 50mm.

3.3 Servomotores

Los servos elejidos para el proyecto son los Towerpro MG995 y sus características son las siguientes:

Longitud	40.6 mm
Anchura	19.7mm
Altura	42.9mm
Peso	55g
Tensión de alimentación	4.8-6V
Par (a 4.8V)	10kg/cm
Velocidad (a 4.8V)	60grados/0.2segundos
Precio	4.5 €

Tabla 3: Características de los servos empleados

Se han escogido estos servos por su tamaño, su precio y porque cumplen con el par mínimo necesario, pero el mayor inconveniente que tienen es la velocidad, sería deseable que fueran más rápidos, ya que una de las principales características de los robots delta es su velocidad.

3.4 Electroimán

El electromián elejido para el proyecto es un electroimán solenoidal y sus características son las siguientes:

Altura	15 mm
Diámetro	20mm
Tensión de alimentación	12V
Corriente máxima	300mA
Peso levantado	hasta 2.5 Kg
Precio	3 €

Tabla 4: Características del electroimán empleado

Se ha escogido este electroimán principalmente por su tamaño y porque cumple el resto de características exigidas.

Para manejar el electroimán se ha implementado un circuito de disparo con un transistor BJT BD139 y una resistencia de base de 220 Ohmios.
Características del transistor:

Ganancia de corriente	15 mm
Corriente máxima por el colector (Ic)	20mm
Tensión máxima de colector a emisor (Vce)	12V
Precio	0.48 €

Tabla 5: Características del transtitor BJT BD139

Figura 22: BD139

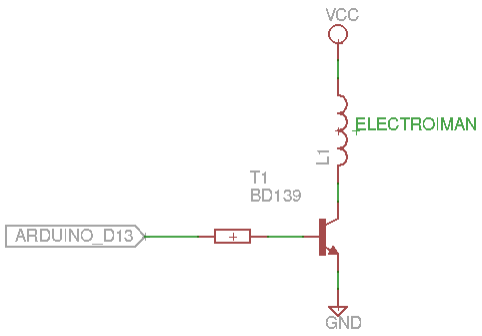


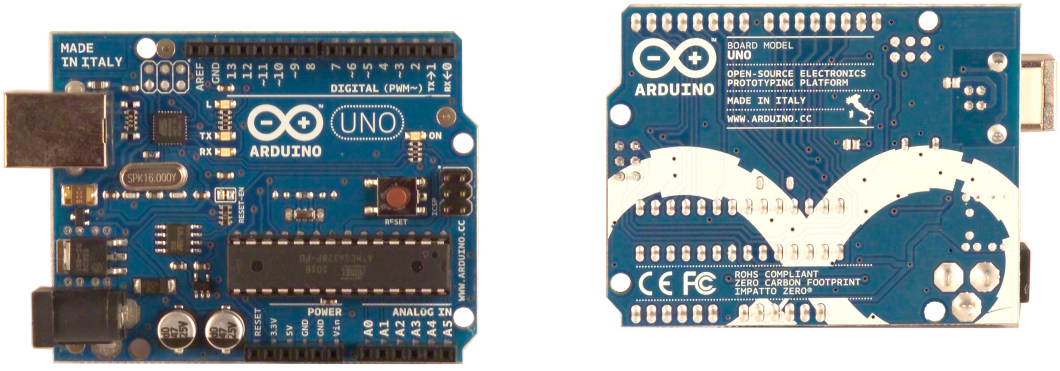
Figura 23: Circuito de disparo del electroimán

3.5 Arduino Uno

El microcontroladore escogido para el proyecto es Arduino Uno y sus características son las siguientes:

Microcontrolador	ATmega328
Tensión de salida	5V
Tensión de alimentación	7-12V
Limites de alimentación	6-20V
E/S Digitales	14 (6 PWM)
Entradas analógicas	6
Corriente por pines E/S	40mA
Memoria flash	32KB
Velocidad de reloj	16MHz
Precio	20 €

Tabla 6: Características de Arduino Uno



(a) Parte delantera (b) Parte trasera

Figura 24: Arduino Uno

Se ha escogido este microcontrolador por la sencillez de manejo, porque cumple con las especificaciones de potencia del proyecto, por tener un entorno de desarrollo multiplataforma, ser de código abierto y por tener las bibliotecas para gestionar la cámara para la visión artificial.

3.6 Pixy CMU5 Camera

Para resolver el problema de la visión artificial se ha escogido la Pixy CMU5 Camera.

Pixy es una colaboración entre el Carnegie Mellon Robotics Institute y Charmed Labs. Viene de un largo legado de CMUcams, pero realmente empezó en kickstarter. Empezó a distribuirse en marzo de 2014 y se ha convertido en es sistema de visión artificial mas popular de la historia. Pixy fue fundada exclusivamente a través de sus ventas.

Características:

- Sistema de visión artificial, rápido, fácil de usar, low cost
- Aprende a detectar objetos que se le enseña
- Salida procesada a 50 fotogramas por segundo
- Fácil de conectar a arduino y a otros microcontroladores
- Provista de todas las bibliotecas necesaria para arduino y otros microcontroladores
- Soporta C/C++ y Python
- Comunicación a través de varia interfaces: SPI, I2C, UART, USB, o salida digital/analógica
- Herramienta de configuración multiplataforma
- Todo el software y firmware es de código abierto, licenciado bajo GNU
- Provisto de toda la información del hardware, esquemáticos, lista de materiales, diseño de la PCB...
- Precio: 60 €

Figura 25: Pixy CMU5 Camera

3.7 Ordenador

El ordenador usado en el proyecto ha sido el PackardBell easynote tn36
Características:

- Procesador: Intel Pentium Dual Core T3400, 2.16 GHz
- Memoria RAM: 4 GB
- Disco duro: 320 GB
- Pantalla: 15.6″, panorámica
- Tarjeta gráfica: Intel Graphics Media Accelerator 4500
- Precio 400 €

Figura 26: Packardbell easynote tn36

4 Software

4.1 Qt creator

Con Qt es posible desarrollar aplicaciones multiplataforma que pueden ser fácilmente compiladas para distintas arquitecturas y sistemas operativos sin grandes modificaciones del código fuente.

Figura 27: Logo de Qt

4.2 Pixymon

Pixymon es la herramienta de configuración para Pixy, con pixymon se puede:

- Enseñar los objetos a detectar
- Cambiar brillo contraste y otros parámetros de la imagen
- Configurar el número máximo de objetos totales y del mismo tipo a detectar
- Cambiar el tipo de interfaz de salida

4.3 Arduino IDE

Es el entorno de desarrollo de Arduino, desde él se pueden crear y flashear sketches en las diferentes placas arduino, tamabién permite crear y cargar bibliotecas, así como hacer un pequeño debug con el

monitor serie. Arduino IDE viene bien nutrido de una gran variedad de sketches de ejemplo que nos ayudan a comenzar nuestros propios sketches.

Figura 28: Logo de Arduino

5 Etapas del proyecto

1. Diseño, fabricación y montaje del robot en si
2. Configuración y programación de la parte de visión artificial
3. Programación de la interfaz de usuario, cálculo de la cinemática del robot y comunicación entre ordenador y arduino, con Qt
4. Programación del control de los actuadores en arduino
5. Programación de una demo donde se muestren las posibilidades del proyecto

A continuación se expone el diagrama de bloques del proyecto para hacerse una idea de las interconexiones de cada una de las partes del proyecto

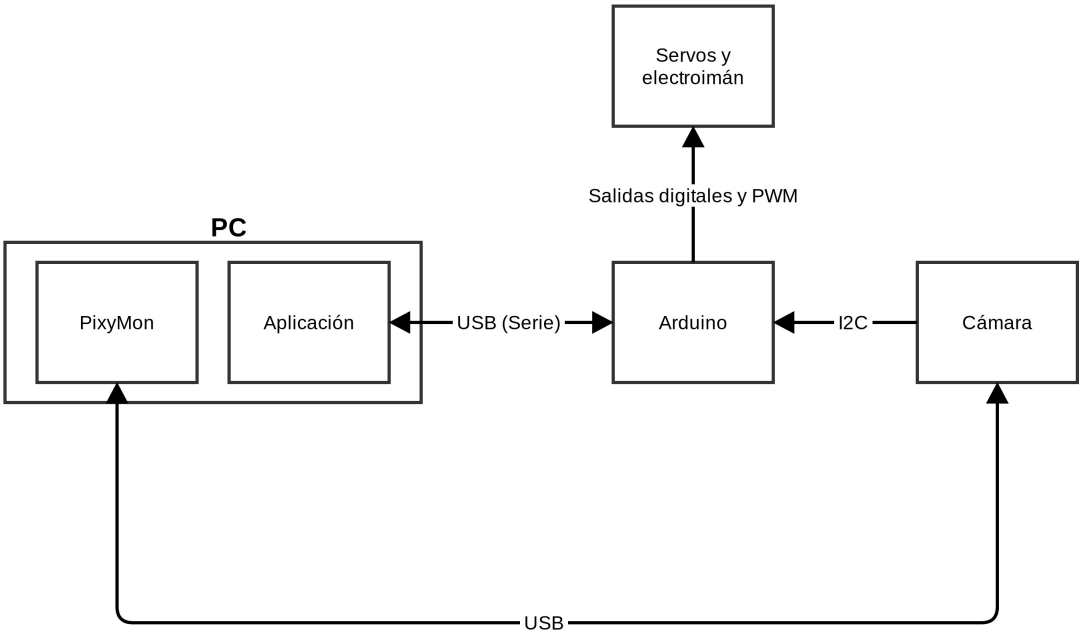


Figura 29: Diagrama de bloques del proyecto

5.1 Diseño, fabricación y montaje del robot

Se hicieron dos modelos de robot delta, el primero más rudimentario, hecho de madera y plastico todo cortado a mano, no era funcional ya que la cinemática calculada no se correspondía en absoluto con la realidad, por lo que se descartó y se hizo un segundo modelo.

Figura 30: Foto del primer modelo de robot delta

Para el segundo modelo se utilizó el diseño descrito en el apartado de hardware, para su fabricación se utilizó una impresora 3D, el ensamblado de las piezas se hizo con tuercas, tornillos y barilla roscada de 3mm. El resultado es muy satisfactorio y los cálculos de la cinemática se corresponden casi al 100% con la realidad, salvo por algunos desvíos, pero no se ha de olvidar que es un modelo casero e impreso con una impresora 3D, no ha habido ningún proceso de fabricación industrial profesional detrás.

El resultado puede verse en la figura 21.

5.2 Configuración y programación de la parte de visión artificial

Todo el desarrollo se hizo en ubuntu 14.04 LTS x64-bit

5.2.1 Instalación de PixyMon

Primero hay que instalar las dependencias para poder compilar PixyMon para ello, abrir un terminal y escribr lo siguiente:

```
sudo apt-get install git
sudo apt-get install libusb-1.0-0-dev
sudo apt-get install qt4-dev-tools
sudo apt-get install qt4-qmake
sudo apt-get install qt4-default
sudo apt-get install g++
```

Una vez instaladas las dependencias hay que descargar y compilar PixyMon

```
git clone https://github.com/charmedlabs/pixy.git
cd pixy/scripts
./build_pixymon_src.sh
```

Y añadir permisos para interfaz USB a Pixy

```
cd ../src/host/linux/
sudo cp pixy.rules /etc/udev/rules.d/
```

Una vez hecho esto ya se puede iniciar PixyMon, para ello en el terminal escribir:

```
cd ../../../build/pixymon/bin/
./PixyMon
```

5.2.2 Actualizar el firmware de Pixy

Para poder comezar a usar Pixy es necesario actualizarle el firmware, eso se hace desde PixyMon siguiendo los siguientes pasos.

Antes que nada hay que asegurarse que:

1. PixyMon está funcionando
2. Se ha descargado el firmware, se pude descargar desde http://cmucam.org/projects/cmucam5/wiki/Latest_release

Una vez cumplidos los requisitos de anteriores:

1. Asegurarse que Pixy no está enchufada por USB
2. Enchufar el cable USB que se conectará a la cámara al PC
3. Mantener apretado el botón blanco de Pixy. Mientras se mantiene apretado el botón conectar el cable USB a Pixy. Una vez conectado el cable se puede soltar el botón

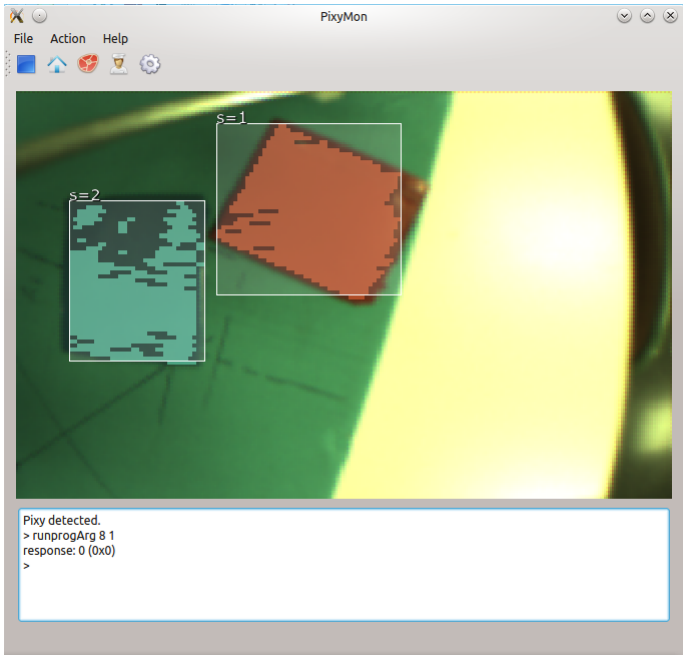
Enchufar el cable USB mientras se mantiene apretado el botón, pone a Pixy en modo actualización de firmware.

En la pantalla de estado de PixyMon (en la parte de abajo de la ventana principal de PixyMon) se debería ver el mensaje "Pixy programming state detected".

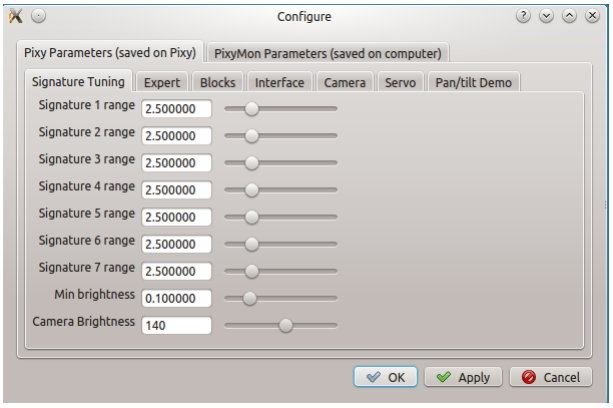
Despues saltará una ventana pidiendo que se seleccione el firmware que se ha descargado (es un archivo .hex), se selecciona y se pulsa OK. Pixy cogerá el programa y lo escribirá en la memoria flash. Una vez terminado se debería ver el mensaje “done!” en la ventana de estatus, lo que quiere decir que el nuevo firmware empezará a funcionar.

5.2.3 Configuración

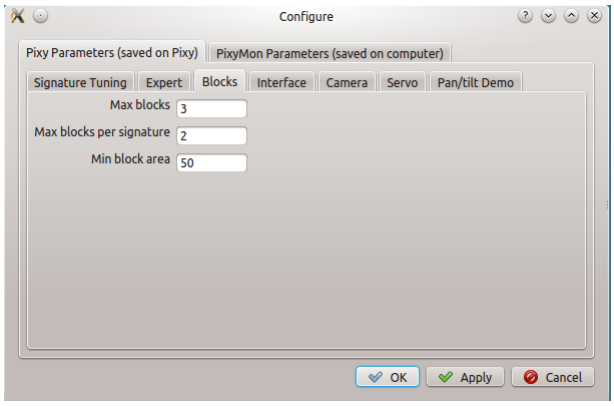
En concreto en proyecto se configuraron los objetos a detectar, el brillo de la cámara, para que la captura fuera óptima y evitar en la medida de lo posible las falsas detecciones, y la interfaz de salida se cambió de SPI a I2C, ya que el SPI interfería en la comunicación entre el PC y el Arduino.



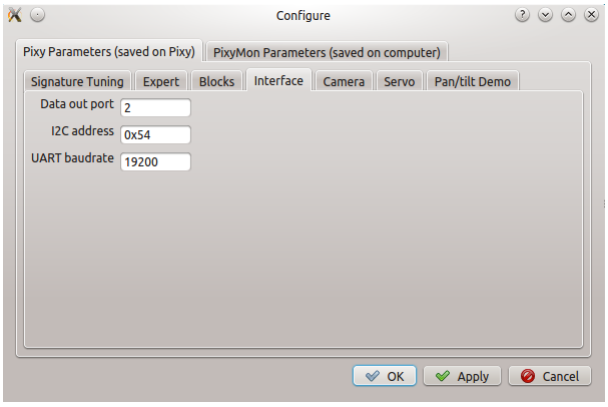
(a) Página principal de pixymon



(b) Pestaña de configuración del brillo



(c) Pestaña de configuración de los objetos a detectar



(d) Pestaña de configuración de la interfaz de salida

Figura 31: Capturas de pantalla de pixymon

Al cambiar la intefaz de salida también se cambió el conexionado de la cámara al arduino, en lugar de gastar el cable que viene con la cámara

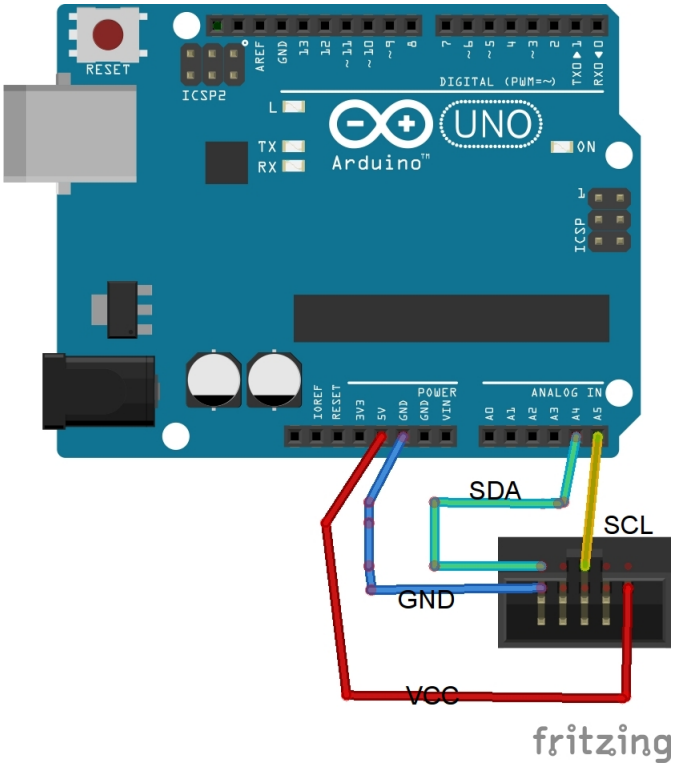


Figura 32: Conexionado Pixy-Arduino

5.2.4 Instalación de Arduino IDE

Para instalar el IDE de Arduino solo es necesaria la siguiente instrucción:

```
sudo apt-get install arduino
```

con esta instrcción Arduino IDE se instalará y configurará para poder iniciarlo.

5.2.5 Importar la bibliotecas de Pixy

Para importar las bibliotecas de Pixy para Arduino hay que seguir los siguientes pasos:

1. Descargar la biblioteca "arduino_pixy-x.y.z.zip" que se puede descargar desde: http://cmucam.org/attachments/download/1054/arduino_pixy-0.1.3.zip
2. En el IDE de Arduino ir a la pestaña de Sketch → Importar biblioteca → Añadir biblioteca.. y se selecciona el .zip con la biblioteca que se ha descargado en paso anterior

Con esto se han importado las bibliotecas y unos cuantos ejemplos de uso de las mismas.

5.2.6 Funciones, variables, instrucciones e includes usados de la librería de Pixy en el proyecto

```
#include <Wire.h>
```

Este include es para gastar el I2C del Arduino.

```
#include <PixyI2C.h>
```

Este include es para comunicar con la cámara a traves del I2C

```
PixyI2C pixy;
```

Pixy I2C es una clase heredada de la clase Block que contiene toda la información de la cámara

```
struct Block
{
  void print()
  {
    char buf[64];
       sprintf(buf, "sig: %d x: %d y: %d width: %d height: %d\n"
, signature, x, y, width, height);
    Serial.print(buf);
    }
  uint16_t signature;
  uint16_t x;
  uint16_t y;
  uint16_t width;
  uint16_t height;
```

La clase block contiene la información de "signature" (número de objeto asignada, puede ir de 1 a 7), la posición X del objeto detectado, la posición Y del objeto detectado, la anchura y la altura del objeto detectado

```
void pixy.init(void);
```

Esta función se pone en el setup y sirve para inicializar la cámara.

```
int pixy.getBlocks(void);
```

Está funcioón devuelve el número de objetos detectados.

5.2.7 Parte del programa de Arduino relacionado con la visión artificial

```
#include <Wire.h>// Incluye la biblioteca del I2C
#include <PixyI2C.h>// Incluye la biblioteca de la camara

//envio de datos de la camara
PixyI2C pixy;//objeto principal de pixy
int inicio = 500;//inicio de trama
int final = 300;//final de trama
String s = "s";
String trama;//trama a enviar
int x, y, signature;
//datos de los objetos detectados en los que se esta interesado
uint16_t blocks;// numero de objetos detectados
```

Esta parte corresponde con la inclusión de las bibliotecas y declaración de las variables necesarias

```
void setup(){
pixy.init();//Inicializar la camara
Serial.begin(115200);//Inicializar la comunicacin serie a 115200 baudios
.
.
```

```
.
}
```

Inicialización de la cámara y del puerto serie para poder enviar las tramas con los datos de las imágenes procesadas

```
void loop(){
.
.
.
//leer de la camara
    blocks = pixy.getBlocks();
//guarda el numero de objetos detectados en la bariable blocks
  if (blocks){// si se ha detectado algo
    j++;
      //como la camara envia 50 fps, para no saturar el arduino se hace
//que procese los datos una de cada 10 veces (cada 200 ms)
    if (j%10==0){
      //sacar los datos de las imagenes
      for (k=0; k<blocks; k++){
//lo hace para cada uno de los objetos detectados
        signature = pixy.blocks[k].signature;
//guarda en la variable signature la asignacin del objeto detectado
//(puede ir de 1 a 7)
        x = map(pixy.blocks[k].x, 0, 300, 0, 100);
//coje la posicin x del objeto y lo pasa de un rango de
//0 a 300 a un rango de 0 a 100
        y = map(pixy.blocks[k].y, 0, 200, 100, 0);
//coje la posicin y del objeto y lo pasa de un rango de
//0 a 200 a un rango de 0 a 100
        trama = s + " " + inicio + " " + signature + " " + x + " "
                + y + " " + final;
// construye una trama con la siguiente estructura: s inicio_de_trama
//asignacion_del_objeto posx posy final_de_trama
        //el inicio y final de trama estan puestos para
//evitar datos erroneos en la recepcion del PC
        Serial.println(trama);// envia la trama por serie al PC
      }
    }
  }
}
```

En este trozo del programa se lee desde la cámara, si ha habido alguna detección suma 1 a j, en el momento en el que j llegue a 10 se extraen los datos de las imágenes y se envían al PC. Si el funcionamiento es correcto y hay objetos que han de ser detectados se procesarán y enviarán datos cada 200ms.

5.3 Programación de la interfaz de usuario, cálculo de la cinemática del robot y comunicación entre ordenador y arduino con Qt

5.3.1 Instalación de Qt creator

La instalación de Qt es algo trivial, simplemete se descarga el programa para el sistema operativos del que se disponga desde aquí https://www.qt.io/download-open-source/#section-2, se ejecuta el

instalador y se siguen los pasos del asistente de instalación.

5.3.2 Creación de una aplicación con GUI en Qt

Una vez esté instalado, se ejecuta Qt creator

1. Se selecciona new project en la venta principal del programa
2. en la pantalla de new project se selecciona Applicaciton → Qt Widgets Application y se hace click en Choose...
3. En la siguiente pantalla se selecciona el nombre y donde se quiere guardar el proyecto y se pulsa Next >
4. Seleccionar el compilador que se quiera utilizar, en este proyecto se ha usado el Desktop Qt 5.4.1 GCC 64bit y se pincha en Next>
5. En esta pantalla se selecciona el nombre de la clase, la clase base, el nombre del fichero de cabeceras y del fuente, se selecciona generate form, se pone nombre al form. En este proyecto se ha dejado todo por defecto. Seguidamente se pincha en Next >
6. En la siguiente pantalla se muestra una lista con ficheros que van a ser generados, se pulsa Finish y el programa creará los ficheros listados.

Si se ejecuta esta aplicación se obtendrá un ventana vacía

5.3.3 Explicación de la solución adoptada

5.3.3.1 Estructura de la aplicación La estructura del programa es la que se describe en diagrama de abajo, está compuesta por una sola clase y distintos métodos y funciones.

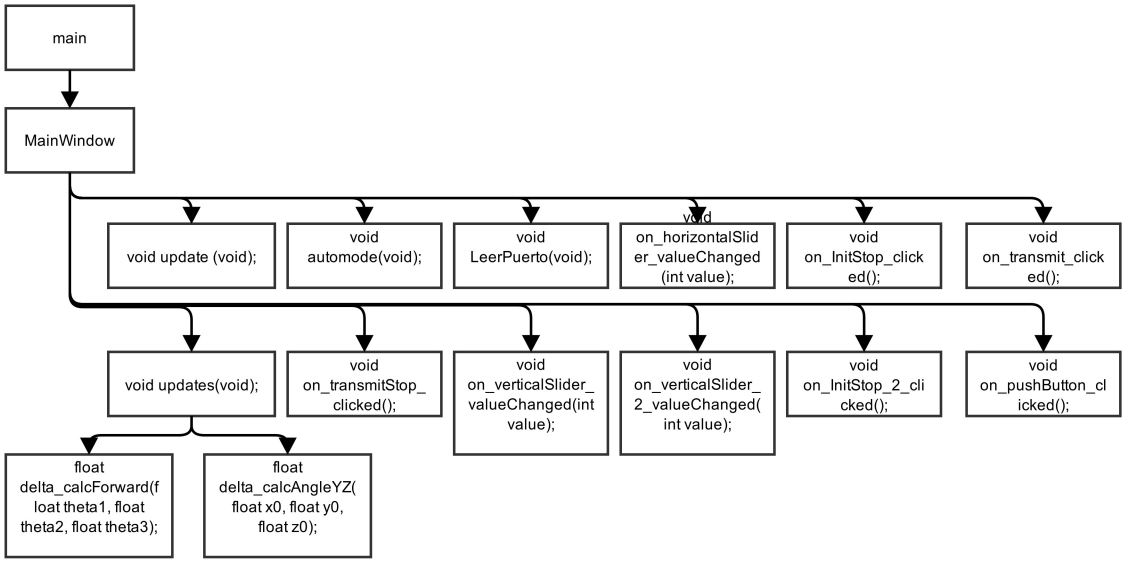


Figura 33: Estructura de la aplicación

- **main**: La función main no es una clase. Es la función de más alto nivel cuyo propósito es iniciar la aplicación, crear y ejecutar la clase MainWindow y reiniciar la aplicación, si MainWindow devuelve un código de reinicio desde la ejecución
- **mainWindow**: Esta es la única clase de la aplicación y contiene todos los métodos y funciones necesarios para el funcionamiento de la misma. Muestra la interfaz de usuario, configura el puerto serie, configura y activa los relojes y conecta todos los slots con sus respectivos signals
- **void update (void)**: Este método se encarga de crear y enviar la trama de control para Arduino. La trama tiene la siguiente estructura: electroimán_ON/OFF número_de_servo signo grados. Está función se activa con un reloj
  - electromán_ON/OFF está compuesto por un byte
    - 0 → electroimán apagado
    - 1 → electroimán encendido
  - número_de_servo está compuesto por un byte y va de 1 a 3, es el encargado de distinguir a que servo van destinados los datos de signo y grados.
  - signo está compuesto por un byte y su función es determinar si el dato de grados es positivo o negativo.
  - grados está compuesto por 3 bytes y son los grados a los que han de moverse los servos (sin tener en cuenta el signo). Grados puede ir desde 000 hasta 180, pero en la práctica en esta aplicación va de 000 a 090, nótese que en grados no se incluye el signo, así pues si se tiene en cuenta el signo la información que se envía a Arduino es de -90 a 90 grados.
- **void updates (void)**: Este método es el encargado de calcular la cinemática, tanto directa como inversa (aunque en el proyecto, solo se usa la inversa) y actualizar los displays con los datos recalculados. Esta función se activa con un reloj
- **void LeerPuerto(void)**: Este método se usa para leer los datos que vienen de Arduino de la visión artificial, como se explica en la sección 5.2.7 y actualizar los displays con los datos leídos. Este método se activa cada vez que hay algo en el buffer del puerto serie
- **void automode(void)**: Este método es una demo para ver todas las secciones del proyecto. Que consiste en que al detectar un pieza en el rango de error, va a por ella y en función del el color la coloca en un lugar o en otro. Esta función se activa con un reloj que a su vez es activado o desactivado por la función void on_pushButton_clicked()
- **void on_horizontalSlider_valueChanged(int value)**: Esta función actualiza el valor de un display con el valor del slider. Esta función se activa cada vez que se cambia el valor del slider
- **void on_InitStop_clicked()**: Esta función da la orden de encender o apagar el electroimán en el modo manual y actualiza el texto del botón. Esta función se activa cada vez que se pulsa el botón InitStop
- **void on_transmit_clicked()**: Esta función abre el puerto serie con la configuración que se hace al iniciar MainWindow y proporciona un mensaje para saber si se ha abierto o no. Está función se activa cada vez que se pulsa el botón transmit
- **void on_transmitStop_clicked()**: Esta función cierra el puerto serie. Esta función se activa cada vez que se aprieta el botón transmitStop
- **void on_verticalSlider_valueChanged(int value)**: Esta función actualiza el valor de un display con el valor del slider. Esta función se activa cada vez que se cambia el valor del slider

- **void on_verticalSlider_2_valueChanged(int value)**: Esta función actualiza el valor de un display con el valor del slider. Esta función se activa cada vez que se cambia el valor del slider
- **void on_InitStop_2_clicked()**: Esta función deja que se pueda actualizar o no, el valor de los displays que muestran la posición y la asignación de los objetos detectados. Esta función se activa cada vez que se pulsa el botón InitStop_2
- **void on_pushButton_clicked()**: Esta función activa o desactiva el modo auto. Esta función se activa cada vez que se pulsa el botón pushButton

5.3.3.2 Diagramas de flujo de la aplicación

Para una mejor explicación a continuación se describen los diagramas de flujo de la aplicación y una captura de pantalla de la interfaz de usuario.

Nótese que no todo el código se describe a en los diagramas, los ficheros de cabecera no aparecen, ya que su único proposito es el de prever la implementación para las funciones.

La función main está compuesta únicamente del documento main.cpp y el diagrama es el siguiente:

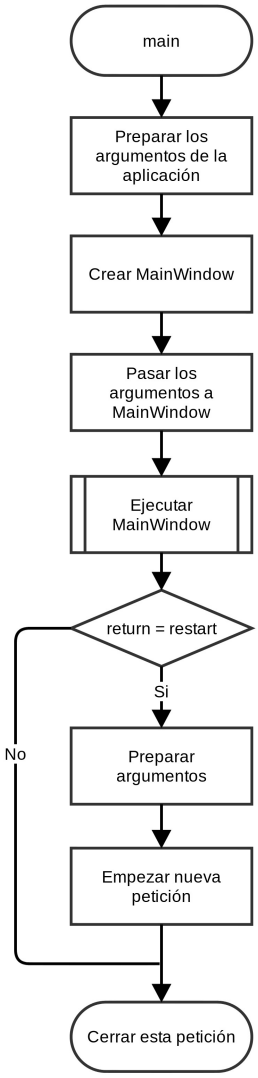


Figura 34: Diagrama de flujo de main

El módulo MainWindow está compuesto por los documentos MainWindo.cpp, MainWindow.h y MainWindow.ui y su diagrama es el siguiente:

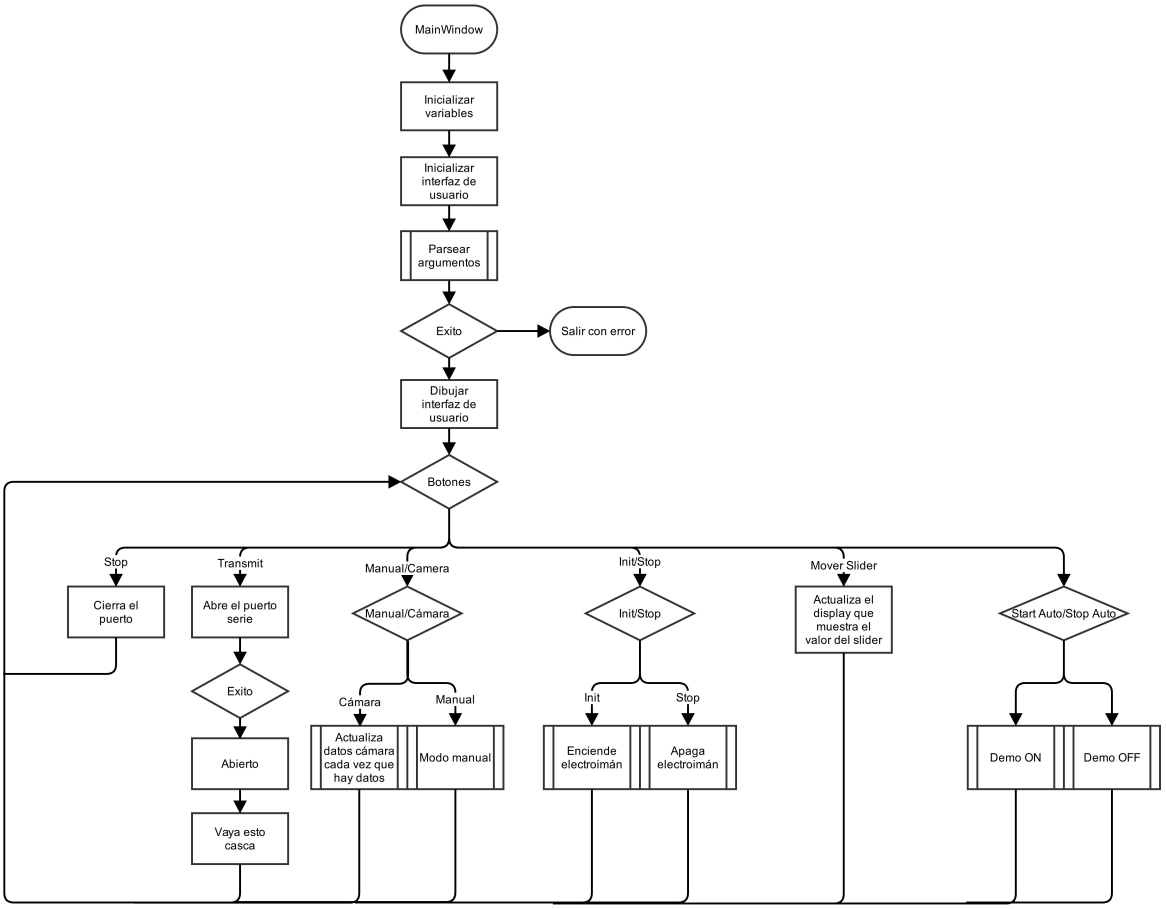


Figura 35: Diagrama de flujo de MainWindow

El *form* de Qt MainWindow.ui es el encargado de crear la interfaz de usuario, cuya captura de pantalla es la siguiente:

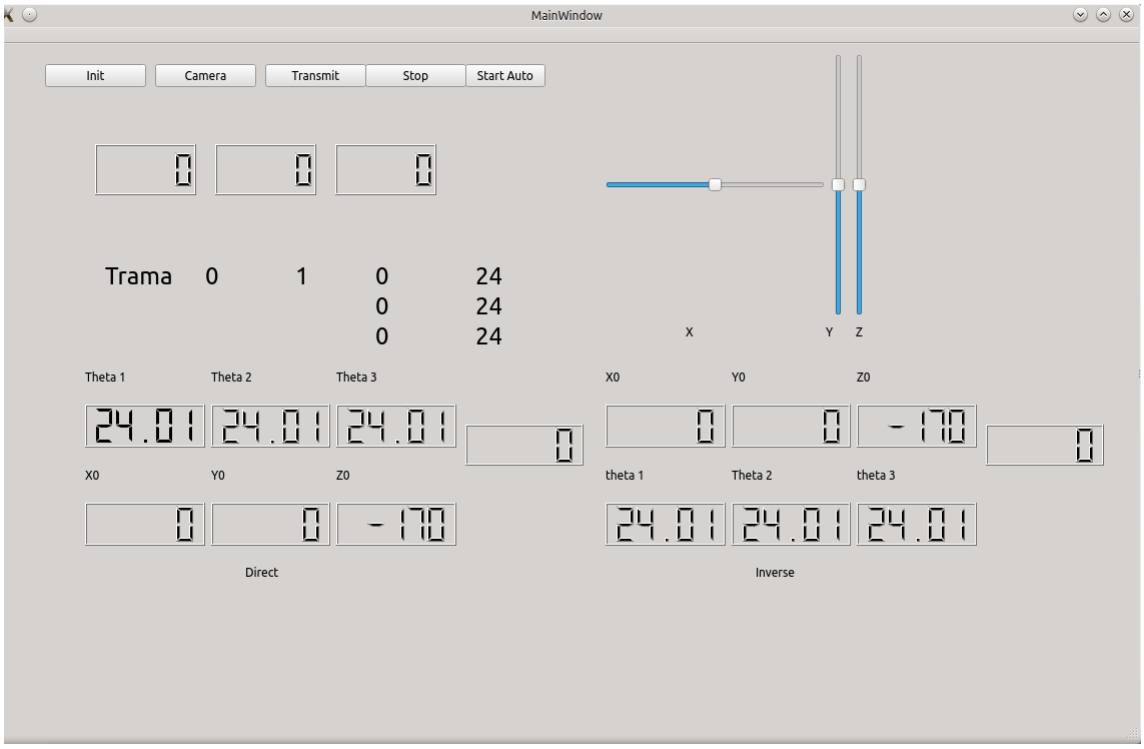


Figura 36: Captura de pantalla de la interfaz de usuario

5.4 Programación del control de los actuadores en Arduino

El control de los actuadores se ha hecho con salidas digitales, para el electroimán y con con PWM a través de la biblioteca servo para los servos.

5.4.1 El electroimán

El electroimán es actuado a través de un circuito de disparo, como se muestra en la figura 23.

5.4.2 Los servomotores

Los servos son unos dispositivos un poco más complejos de controlar ya que se puede controlar la posición y la velocidad. Los servomotores hacen uso de la modulación por ancho de pulsos (PWM) para controlar la dirección o posición (en este proyecto, posición) de los motores de corriente continua. La mayoría trabaja en la frecuencia de los cincuenta hercios, así las señales PWM tendrán un periodo de veinte milisegundos. La electrónica dentro del servomotor responderá al ancho de la señal modulada. Con un ciclo de trabajo de entre 1ms y 2ms.

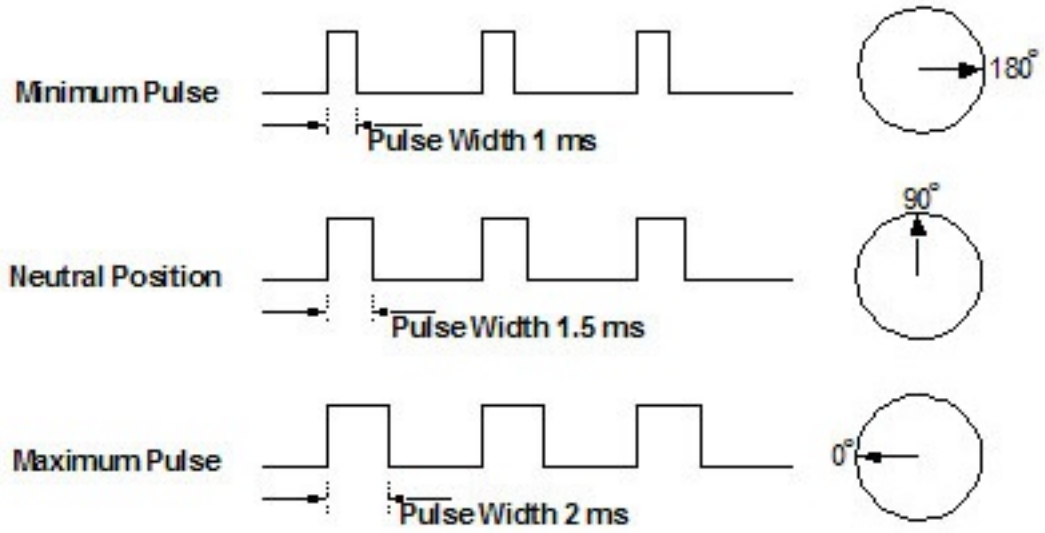


Figura 37: Control de giro del servo

La biblioteca servo nos proporciona un conjunto de instrucciones de alto nivel para el control de los servos. En el proyecto del robot delta se han usado las siguientes partes de la bibliotecas servo:

```
#include <Servo.h>
```

Inculye la biblioteca en el programa

```
Servo servo1,servo2,servo3;
```

Crea tres objetos de servo, uno por cada servo físico

```
servo1.attach(8);
servo2.attach(10);
servo3.attach(11);
```

Asigna cada uno de los objetos de servo creados en el paso anterior a un pin físico del arduino

```
case 1: servo1.write(90-2-Degrees); break;
case 2: servo2.write(90-2-Degrees); break;
case 3: servo3.write(90-2-Degrees); break;
```

Genera un pulso PWM para que el servo se mueva a la posición, en grados, deseada

5.5 Parte del programa de Arduino relacionada con el control de los actuadores

```
#include <Servo.h>// Incluye la biblioteca para el control de los servos
//Servos y electroiman
Servo servo1,servo2,servo3;//objetos de servo
int led = 13;//pin del electroiman
//recepcion de tramas
int StartStop,servo,Degrees,sign;//datos de la trama que viene desde el PC
char lectura2,l2,l3;
static char lectura[20],CStartStop[20],CServo[20],CSign[20],CDegrees[20];
```

Esta parte corresponde con la inclusión de la biblioteca servo y declaración de las variables necesarias.

```
void setup(){
.
.
.
pinMode (led, OUTPUT);// poner el pin del electroiman en modo salida
Serial.begin(115200);//Inicializar la comunicacin serie a 115200 baudios
servo1.attach(8);//asignar el servo 1 a al pin 8
servo2.attach(10);//asignar el servo 2 a al pin 10
servo3.attach(11);//asignar el servo 3 a al pin 11
}
```

En esta sección se inicializan las salidas donde irán conectados los actuadores y el puerto serie para poder recibir las tramas con la información para los actuadores.

```
void loop(){
  if (Serial.available() > 10){// si se ha enviado algo desde el PC
    for (i=0;i<12;i++){//leelo
      lectura2 = Serial.read();
      lectura[i] = lectura2;// y guardalo en la variable lectura
     }
  }
  /////Start Stop////////////
  CStartStop[0]=lectura[0];
  StartStop = atoi(CStartStop);
  ///// N Servo////////////
  CServo[0]=lectura[2];
  servo= atoi(CServo);
  ///// PositiveNeganitve////
  CSign[0]=lectura[4];
  sign=atoi(CSign);
  ///// Grados////////////
  for(i=6;i<9;i++){
    CDegrees[i-6]=lectura[i];
  }
   Degrees=atoi(CDegrees);
  if (sign == 1){
    Degrees = -Degrees;
  }
.
.
.
```

En esta sección se lee el puerto serie si hay alguna trama y se procesan los datos que vienen en forma de *static char* y se pasan a *int*, que es el tipo de variable que espera la función *servo.write(int data);*.

```
.
.
.
if (StartStop == 1){//enciende o apaga el electroman
    digitalWrite(led, HIGH);
```

```
  }
  else {digitalWrite(led, LOW);
}

switch (servo){// mueve los servos a la posicion correspondiente
    case 1: servo1.write(90-2-Degrees); break;
//la posicin es 90 - factor de correccion de error de los servos
- grados enviados desde el PC
    case 2: servo2.write(90-2-Degrees); break;
    case 3: servo3.write(90-2-Degrees); break;
     }
.
.
.
}
```

En esta sección con los datos de la sección anterior se mueven los actuadores como corresponda. Nótese que la posición de los servos se ve modificada ligeramente por un factor de corrección, para compensar el error de los servos cuando Degrees es 0 y que Degrees estará entre -90 y 90 como se explica en la sección 5.3.3.1.

Ahora que todo la programación hecha en Arduino está explicada se mostrará un diagrama de flujo del programa completo.

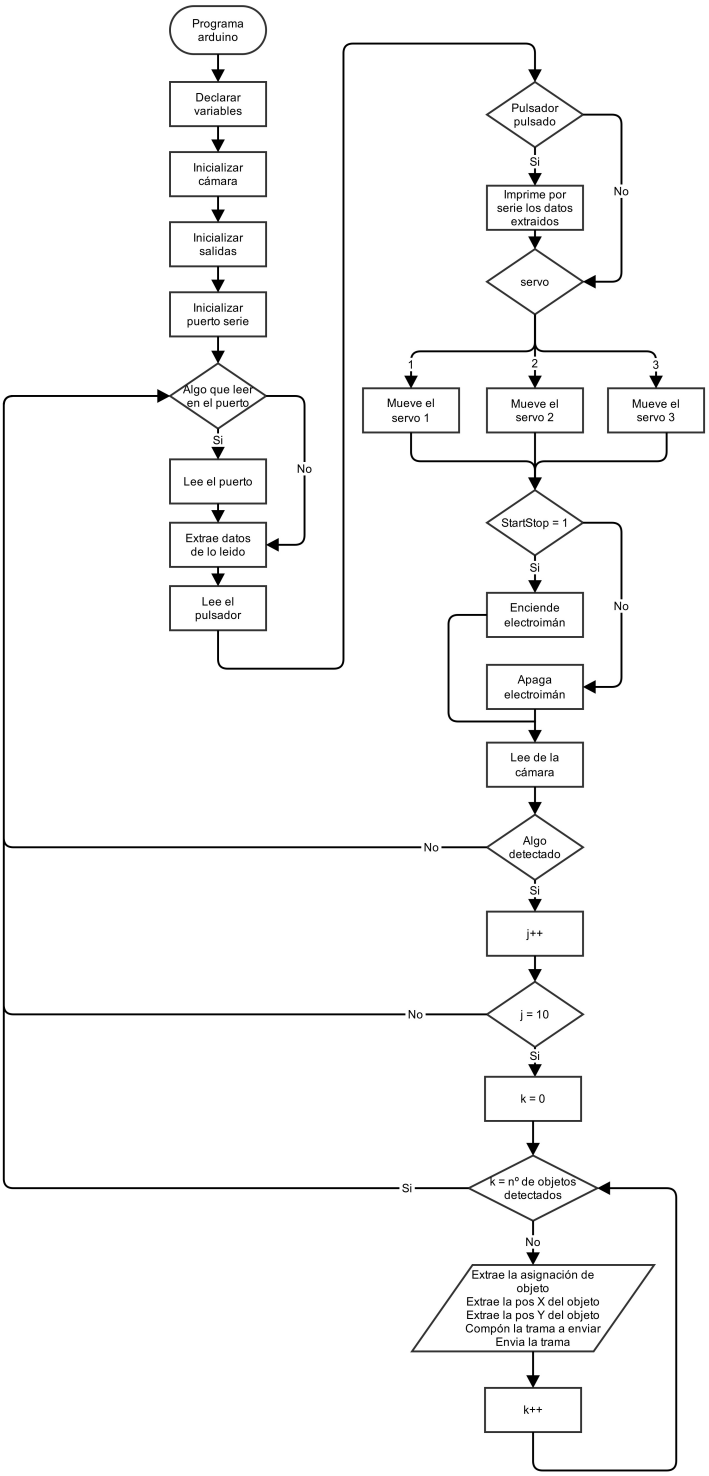


Figura 38: Diagrama de flujo del código de Arduino

5.6 Programación de una demo donde se muestren los posibilidades del proyecto

Esta parte ya se explicó por encima en la sección 5.3.3.1, pero en esta sección se explicará más detalladamente.

La demo consiste en un máquina de estados. En cada estado hará unas acciones y determinará el siguiente estado y se saldrá de la función. Los cada estado se detalla a continuación

- **Estado 0**: Este es estado inicial, aquí robot se coloca en la posición inicial, comprueba que haya alguna pieza que coger, el color y si está en un lugar asignado para la recogida de piezas. Si se cumplen que hay un pieza a la vista y está donde toca, reconfigura el reloj que llama a esta función, pasa al estado 1 y se sale de la función.
- **Estado 1**: En este estado el robot se mueve a la posición de recogida de piezas, pero a unos centímetros por encima de ella, reconfigura el reloj que llama a la función, pasa al estado 2 y se sale de la función.
- **Estado 2**: En este estado el robot baja, manteniendo la posición X e Y hasta tocar la pieza, con el electroimán encendido, pasa al estado 3 y sale de la función.
- **Estado 3**: En este estado el robot vuelve a la posición del estado 1 pero con la pieza cogida, pasa al estado 4 y se sale de la función
- **Estado 4**: En este estado el robot vuelve a la posición inicial y en función del color de la pieza, pasa al estado 8 o al 5 y sale de la función
  - Si el color de la pieza es el de la asignación 1 va al estado 5
  - Si el color de la pieza es el de la asignación 2 va al estado 8. Recuérdese que pueden haber hasta 7 asignaciones diferentes y dichas asignaciones se hacen desde PixyMon
- **Estado 5**: En este estado el robot va a la posición X e Y, manteniendo Z constante, determinada para piezas del color de la asignación 1, pasa al estado 6 y sale de la función
- **Estado 6**: En este estado el robot baja hasta el nivel del "suelo" donde ha de depositarse la pieza, manteniendo X e Y constantes, pasa al estado 7 y se sale de la función
- **Estado 7**: En este estado el robot apaga el electroimán, por lo que ya no está sujetando la pieza, reconfigura el reloj que llama a la función, pasa al estado 0 y sale de la función.
- **Estado 8**: En este estado el robot va a la posición X e Y, manteniendo Z constante, determinada para piezas del color de la asignación 2, pasa al estado 9 y sale de la función
- **Estado 9**: En este estado el robot baja hasta el nivel del "suelo" donde ha de depositarse la pieza, manteniendo X e Y constantes, pasa al estado 10 y se sale de la función
- **Estado 10**: En este estado el robot apaga el electroimán, por lo que ya no está sujetando la pieza, reconfigura el reloj que llama a la función, pasa al estado 0 y sale de la función.

Para una mejor explicación a continuación se muestra un diagrama de flujo del método:

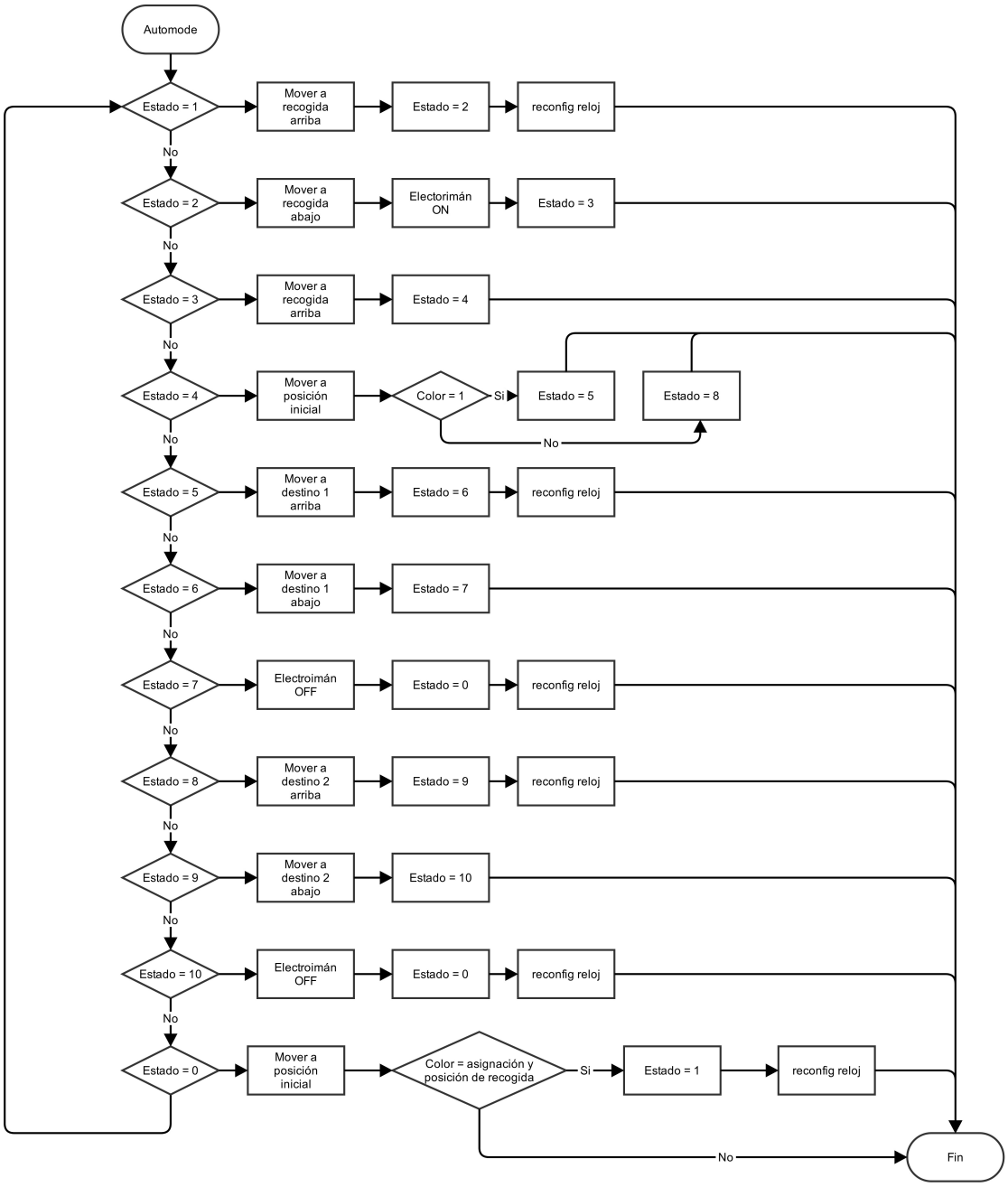


Figura 39: Diagrama de flujo de la demo

6 Conclusiones y trabajo futuro

Teniendo en cuenta que cada vez los robots están más presentes en la industria, debido a que los costes de desarrollo han bajado por la expiración de algunas patentes que solo poseían algunas empresas o universidades, que su rendimiento es mayor cada día y que los sistemas de visión artificial se están popularizando, debido a la potencia de cálculo de los microcontroladores y ordenadores. Es

predecible que en un futuro su presencia va a ser aún mayor.

En este proyecto se pretendía mostrar que con herramientas como Qt es posible crear aplicaciones multiplataforma para ordenadores industriales o sistemas embebidos y todo lo plasmado en este proyecto es extensible a otros sistemas y microcontroladores con unos cambios mínimos.

En el desarrollo de este proyecto se ha requerido profundizar en la cinemática de robots paralelos y la visión artificial y se han mezclado materias tan diversas como la programación y la mecánica. El proyecto ha servido de gran experiencia en el desarrollo de productos industriales dentro del campo de la robótica.

El equipo de desarrollo del proyecto se siente satisfecho con los resultados obtenidos y aunque hay margen de mejora se podría considerar que todos los objetivos han sido cumplidos.

Algunos partes a mejorar o cambiar en un futuro serían:

- **Cambio de actuadores**: Debido a los servos que se han instalado, el robot es lento, en un trabajo futuro sería deseable instalar unos servos más rápidos o incluso prescindir de los servos, ya que no proporcionan ningún feedback a cerca de su posición, e instalar motores de CC o CA con encoders, lo que dificultaría un poco el desarrollo pero proporcionaría una mayor velocidad y un feedback con la posición del motor. También sería deseable aunque en menor relevancia cambiar el electroimán por una bomba de vacío lo que permitiría coger objetos no imantables.
- **Cambio del sistema de visión artificial a OpenCV**: En un periodo temprano del proyecto este iba a ser el sistema de visión artificial, pero debido a la gran cantidad de tiempo que se requería para aprender correctamente OpenCV y OpenTLD se descartó para este proyecto. OpenCV hubiera proporcionado más versatilidad en las opciones de visión artificial y lo que es más importante, no depender un hardware específico, cualquier cámara que se pueda conectar por USB hubiera valido, pero también hubiera sido necesario trabajar con un ordenador mas potente, al menos en cuanto a tarjeta gráfica.

7 Bibliografía

[1] Articulo sobre robots delta http://en.wikipedia.org/wiki/Delta_robot
[2] Página principal de las bibliotecas de visión artificial "opensoruce computer vision" http://opencv.org/
[3] Página principal del proyecto CMU5 camera http://www.cmucam.org/
[4] Página principal del proyecto para desarrollo C++ Qt http://www.qt.io/
[5] Página principal de arduino http://www.arduino.cc/
[6] V. Poppeová, J. Uríček, V. Bulej, P. Šindler: "Delta robots - Robots for high speed manipulation" Technical Gazette 18, 3(2011), 435-445
[7] mzavatasky: "Delta robot kinematics" Trossen Robotics 27/07/2009 http://forums.trossenrobotics.com/tutorials/introduction-129/delta-robot-kinematics-3276/
[8] Pagina con diseños para impresión en 3D http://www.thingiverse.com/
[9] Diseño hecho por i-make-robots y publicado el 5 de marzo de 2014 http://www.thingiverse.com/thing:263500

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

2. Pliego de condiciones

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

Índice

1 Especificaciones

1.1 Especificaciones del software

El software deberá desarrollarse cumpliendo con las siguientes especificaciones:

- El software utilizado debe permitir ser compilado para distintos sistemas operativos.
- Todo el código será desarrollado mediante software de código libre
- La aplicación debe de ser capaz de comunicarse con el arduino

1.2 Especificaciones de hardware

El hardware empleado en el desarrollo del proyecto debe cumplir las siguientes condiciones:

- El hardware ha de ser libre en la medida de lo posible (los actuadores, como los servos..., no son libres, pero el mercado ofrece muchos modelos y marcas diferentes)
- Los dispositivos empleados han de ser capaces de comandar actuadores
- El ordenador empleado deberá disponer de varios puertos USB

1.3 Especificaciones de la plataforma de desarrollo

1.3.1 Software necesario

Para poder crear la aplicación es necesario disponer del siguiente software:

- Qt creator
- Arduino IDE
- Bibliotecas de Pixy para Arduino
- PixyMon
- Sistema operativo Linux 32 o 64 bit (empleado Kubuntu 14.04 x64)

1.3.2 Hardware necesario

Para el desarrollo del proyecto es necesario cierto hardware y con unos requisitos mínimos

- Pixy CMU5 Camera
- Ordenador que cumplan con las especificaciones necesarias para utilizar el sistema operativo, todo software necesario y al menos dos puertos USB
- Fuente de alimentación, para alimentación de los servos y el electroimán
- Arduino Uno o superior
- Robot delta completo

2 Robot delta

2.1 Enumeración de las piezas

El robot está compuesto por las siguientes piezas:

- 12x articulaciones universales
- 3x bíceps como unión entre los actuadores y las uniones universales
- 6x codos para la unión entre las articulaciones universales y los bíceps
- 3x trozos de base
- 1x efector final
- Tuercas, tornillos y varilla roscada
- 3x Servos TowerPro MG995
- 1x electroimán solenoidal

2.2 Montaje de la base

Las tres piezas que forman la base encajan entre ellas formando una base triangular con tres ranuras para colocalar los servos.

2.3 Montaje de un brazo

Cada brazo está compuesto por:

- 4 articulaciones universales
- 1 bíceps
- 2 codos
- 1 servo
- 1 tornillo y una tuerca de 3mm de grosor, para sujetar el bíceps a la articulación rotatoria del servo
- 2 trozos de varilla de 3mm de grosor roscada, de unión entre dos articulaciones universales
- 4 tuercas y tornillos de 3mm de grosor, de union entre los codos y las articulaciones universales
- 2 trozos de varilla roscada y 2 tuercas de 3mm de grosor, para sujetar los codos a el bíceps y el efector final respectivamente

Ahora se explicará como montar un brazo:

1. Cortar dos trozos largos (en el proyecto de 15 cm) de varilla roscada, el parámetro cinemático de *re*, variará en función de la longitud de esta varilla.
2. Enroscar esos trozos de varilla en los agujeros no pasantes de las articulaciones universales.
3. Cortar dos trozos cortos (la longitud necesaria para que llegue de lado a lado de la parte interior del codo y las tuercas queden encajadas en los agujeros correspondientes).

4. Unir el bíceps, por el extremo cerrado, con el codo y sujetar con la varilla cortada en el paso anterior y dos tuercas.
5. Unir un extremo de cada una de las varillas obtenidas en el paso 2 al codo con el bíceps obtenido en el paso 4 con un tornillo y una tuerca cada una.
6. Repetir el paso 3
7. Unir una parte del efector final igual que se ha hecho con el bíceps
8. Unir los extremos restantes de las varillas del paso 2 con el codo con efector final del paso 7
9. Poner el servo en una posición conocida, es recomendable ponerlo a 90 grados, ya que esa será la posición por defecto del servo en condiciones de trabajo
10. Unir el extremo abierto del bíceps con la articulación rotatoria del servo, ajustar de forma que quede perpendicular al servo (si se ha ajustado en el paso anterior el servo a 90 grados) y asegurar con un tornillo y una tuerca

Repetir cada uno de los pasos anteriores para cada uno de los brazos.

2.4 Union de los brazos a la base

Una vez que todos los brazos están montados y unidos al efector final hay que introducir un extremo de los servos en cada una de las ranuras de la base. De forma que los bíceps queden mirando hacia afuera.

Ahora que todo está montado debería quedar algo parecido a la figura 21 de la memoria.

2.5 Montaje del electroimán

Hay que tener en cuenta que el electroimán va a ir montado en el efector final, y el circuito de disparo va a ir en la base, por lo que hay que dejarle cable suficiente para que no se produzcan tirones en ninguna de las posible posiciones del robot. Dicho esto solo queda colocar en el electroimán en el efector final, en este proyecto se ha sujetado con pegamento.

3 Alimentación

Para que la placa Arduino funcione debe estar alimentada en un rango de 7 a 12V, los cuales se suministran desde el USB del ordenador.

La cámara puede ir alimentada a través del USB al ordenado a través del arduino conectado a la salida de 5V.

Los servos van alimentados a una tensión de 5V, y en función de la velocidad a la que vayan hará falta en 100mA y 0.5 A, por lo que se conectan a través de una fuente de alimentación.

El electroimán se alimenta a 12V y tiene un consumo de 300 mA por lo que va conectado a una fuente de alimentación.

4 Puesta en marcha

Para poner en marcha el equipo es recomendable aunque no obligatorio seguir los siguientes pasos, aunque si que hay algunos pasos que han de ser seguidos en el orden que se indica:

Teniendo en cuenta el ordenador ya está enchufado y dentro del sistema operativo

1. Enchufar el arduino al ordenador a través de un cable USB
2. Enchufar la cámara al ordenador a través de un cable USB
3. Ejecutar PixyMon y configurar las asignaciones. (este paso solo es necesario en caso de que se quieran cambiar las asignaciones de otras sesiones de trabajo anteriores)
4. Ejecutar la aplicación de control
5. Enchufar la fuente de alimentación
6. Pulsar el botón "transmit" de la aplicación para establecer la comunicación entre el arduino y el ordenador.

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

3. Presupuesto

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

Índice

1 Coste del material

Para el coste del material se tiene en cuenta que el cliente tiene un ordenador para ejecutar la aplicación, por lo que no se incluye en el presupuesto, tampoco se incluyen en el presupuesto el precio de las licencias del software, ya que son gratuitas.

Concepto	**Precio unitario (€)**	**Unidades**	**Total (€)**
Impresión del robot	40	1	40
Servomotores TowerPro MG995	4.5	3	13.5
Electroimán	3	1	3
Arduino Uno	20	1	20
Pixy CMU5 camera	60	1	60
Varilla roscada, tornillos y tuercas	3	1	3
Total			139.5

Coste total de los materiales: 139.5 €

2 Coste de la mano de obra

Concepto	Unidades	Coste unitario (€)	Cantidad	Total (€)
Diseño del proyecto	h	15	40	600
Progrmación	h	15	100	1500
Montaje del robot	h	15	4	60
Testeo	h	15	25	375
Documentación	h	15	35	525
Total				3060

Coste total de la mano de obra: 3060 €

3 Total proyecto

Concepto	**Coste (€)**
Coste de los materiales	139.5
Coste de la mano de obra	3060
Total	3199.5

Coste total del proyecto: 3199.5 €

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

Apéndice 1. Documentación del código de la aplicación

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

En este apéndice se encuentra la documentación del código de la aplicación. La documentación ha sido generada por Doxygen 1.8.6 a partir de los archivos de código fuente de la aplicación.

Contents

Chapter 1

Namespace Index

1.1 Namespace List

Here is a list of all namespaces with brief descriptions:

Chapter 2

Hierarchical Index

2.1 Class Hierarchy

This inheritance list is sorted roughly, but not completely, alphabetically:

Chapter 3

Class Index

3.1 Class List

Here are the classes, structs, unions and interfaces with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Namespace Documentation

5.1 Ui Namespace Reference

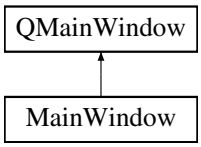
Chapter 6

Class Documentation

6.1 MainWindow Class Reference

`#include <mainwindow.h>`

Inheritance diagram for MainWindow:



Public Slots

- void **update** (void)

  MainWindow::update *(*p. *13) método encargado de la composición y envío de la trama para control de los actuadores en arduino y actualizar la trama en la GUI.*
- void **updates** (void)

  MainWindow::updates *(*p. *13) método que llama las funciones de cálculo de cinemática y actualiza los valores de unos displays con los resultados.*
- void **LeerPuerto** (void)

  MainWindow::LeerPuerto *(*p. *12) lee la trama que viene del arduino con los datos de la cámara y en caso de que la trama recivida sea correcta y se quiera que se actualicen los datos actualiza uns displays.*
- void **automode** (void)

  MainWindow::automode *(*p. *12) método que sirve de demostración.*

Public Member Functions

- **MainWindow** (QWidget ∗parent=0)

  MainWindow::MainWindow *(*p. *12) Única clase de la aplicación. Dibuja la gui, se encarga de la selección de botones, configura todas las signals y slots e inicializa los temporizadores.*
- ∼**MainWindow** ()

  MainWindow::∼MainWindow *(*p. *12).*
- float **delta_calcForward** (float theta1, float theta2, float theta3)

  MainWindow::delta_calcForward *(*p. *12) función que calcula la cinemática inversa.*
- float **delta_calcAngleYZ** (float x0, float y0, float z0)

  MainWindow::delta_calcAngleYZ *(*p. *12) Función que calcula la cinemñatica inversa (solo uno de los ángulos)*

6.1.1 Constructor & Destructor Documentation

6.1.1.1 MainWindow::MainWindow (QWidget * *parent* = 0) [explicit]

MainWindow::MainWindow (p. 12) Única clase de la aplicación. Dibuja la gui, se encarga de la selección de botones, configura todas las signals y slots e inicializa los temporizadores.

Parameters

parent	

6.1.1.2 MainWindow::~MainWindow ( )

MainWindow::~MainWindow (p. 12).

6.1.2 Member Function Documentation

6.1.2.1 void MainWindow::automode (void) [slot]

MainWindow::automode (p. 12) método que sirve de demostración.

6.1.2.2 float MainWindow::delta_calcAngleYZ (float *x0,* float *y0,* float *z0*)

MainWindow::delta_calcAngleYZ (p. 12) Función que calcula la cinemñatica inversa (solo uno de los ángulos)

Parameters

x0	posición X del elector final
y0	posición Y del efector final
z0	posición Z del efector final

Returns

ángulo que ha de tomar el servo

6.1.2.3 float MainWindow::delta_calcForward (float *theta1,* float *theta2,* float *theta3*)

MainWindow::delta_calcForward (p. 12) función que calcula la cinemática inversa.

Parameters

theta1	Ángulo del servo 1
theta2	Ángulo del servo 2
theta3	Ángulo del servo 3

Returns

este valor dice si es una posición posible o no

6.1.2.4 void MainWindow::LeerPuerto (void) [slot]

MainWindow::LeerPuerto (p. 12) lee la trama que viene del arduino con los datos de la cámara y en caso de que la trama recivida sea correcta y se quiera que se actualicen los datos actualiza uns displays.

6.1.2.5 void MainWindow::update (void) [slot]

MainWindow::update (p. 13) método encargado de la composición y envío de la trama para control de los actuadores en arduino y actualizar la trama en la GUI.

6.1.2.6 void MainWindow::updates (void) [slot]

MainWindow::updates (p. 13) método que llama las funciones de cálculo de cinemática y actualiza los valores de unos displays con los resultados.

The documentation for this class was generated from the following files:

- /home/alejandro/Dropbox/Delta robot/sliderDeltaRobot/**mainwindow.h**
- /home/alejandro/Dropbox/Delta robot/sliderDeltaRobot/**mainwindow.cpp**

Chapter 7

File Documentation

7.1 /home/alejandro/Dropbox/Delta robot/sliderDeltaRobot/main.cpp File Reference

```
#include "mainwindow.h"
#include <QApplication>
```

Functions

- int **main** (int argc, char ∗argv[])

7.1.1 Function Documentation

7.1.1.1 int main (int *argc,* char ∗ *argv[]*)

7.2 /home/alejandro/Dropbox/Delta robot/sliderDeltaRobot/mainwindow.cpp File Reference

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <stdio.h>
```

Variables

- char **position1** [1000]
- char **position2** [1000]
- char **position3** [1000]
- char **trama** [1000]
- char **ini** [1000]
- char **serv** [1000]
- int **check**

7.2.1 Variable Documentation

7.2.1.1 int check

7.2.1.2 char ini[1000]

7.2.1.3 char position1[1000]

7.2.1.4 char position2[1000]

7.2.1.5 char position3[1000]

7.2.1.6 char serv[1000]

7.2.1.7 char trama[1000]

7.3 /home/alejandro/Dropbox/Delta robot/sliderDeltaRobot/mainwindow.h File Reference

```
#include <QMainWindow>
#include <QTimer>
#include <QSerialPort>
#include <stdio.h>
#include <math.h>
```

Classes

- class **MainWindow**

Namespaces

- **Ui**

Macros

- #define **BUFFMAX** 1000

7.3.1 Macro Definition Documentation

7.3.1.1 #define BUFFMAX 1000

Index

UNIVERSITAT POLITÈCNICA DE VALÈNCIA

Escuela Técnica Superior de Ingeniería del Diseño

Univeristat Poltècnica de València

Escuela Técnica Superior de Ingeniería del Diseño

Grado en Ingeniería Electrónica Industrial y Automática

Trabajo de final de grado

SISTEMA DE CONTROL DE ROBOT DELTA BASADO EN VISIÓN ARTIFICIAL

Apéndice 2. Código

Alumno:
Alejandro Beltrán Nova

Director del proyecto:
Houcine Hassan Mohamed

Junio 2015

En este apéndice se encuentra todo el código fuente del proyecto, el de la aplicación y el de Arduino. La aplicación ha sido escrita en C++ con Qt creator y el código de Arduino ha sido escrito en el lenguaje propio de Arduino con el entorno de desarroyo Arduino IDE.

Contents

1 main.cpp

```
#include "mainwindow.h"
#include <QApplication>

int main(int argc, char *argv[])
{
    QApplication a(argc, argv);
    MainWindow w;
    w.show();

    return a.exec();
}
```

2 programaarduino.ino

```
#include <Wire.h>// Incluye la biblioteca del I2C
#include <PixyI2C.h>// Incluye la biblioteca de la camara
#include <Servo.h>// Incluye la biblioteca para el control de los servos
//envio de datos de la camara
PixyI2C pixy;//objeto principal de pixy
int inicio = 500;//inicio de trama
int final = 300;//final de trama
String s = "s";
String trama;//trama a enviar
int x, y, signature;//datos de los objetos detectados en los que se et
    interesado
uint16_t blocks;// numero de objetos detectados
//servos y electroiman
Servo servo1,servo2,servo3;//objetos de servo
int led = 13;//pin del electroiman
int button = 2;//botn para debug
int button_state = LOW;
//misc
int i, j=0, k;
//recepcion de tramas
int StartStop,servo,Degrees,sign;// datos de la trama que viene desde el PC
char lectura2,l2,l3;
static char lectura[20],CStartStop[20],CServo[20],CSign[20],CDegrees[20];


void setup(){
  pixy.init();//Inicializar la camara
  pinMode (led, OUTPUT);// poner el pin del electroiman en modo salida
  Serial.begin(115200);//Inicializar la comunicacin serie a 115200 baudios
  servo1.attach(8);//asignar el servo 1 a al pin 8
  servo2.attach(10);//asignar el servo 2 a al pin 10
  servo3.attach(11);//asignar el servo 3 a al pin 11
}

void loop(){
  if (Serial.available() > 10){// si se ha enviado algo desde el PC
    for (i=0;i<12;i++){//leelo
      lectura2 = Serial.read();
      lectura[i] = lectura2;// y guardalo en la variable lectura
    }
  }
  /////Start Stop////////////
  CStartStop[0]=lectura[0];
  StartStop = atoi(CStartStop);
  ///// N Servo//////////////
  CServo[0]=lectura[2];
  servo= atoi(CServo);
  ///// PositiveNeganitve////
  CSign[0]=lectura[4];
  sign=atoi(CSign);
  ///// Grados//////////////
  for(i=6;i<9;i++){
    CDegrees[i-6]=lectura[i];
  }
  Degrees=atoi(CDegrees);
```

```
  if (sign == 1){
    Degrees = -Degrees;
  }

  //solo para debug
  button_state = digitalRead(button);
  if (button_state == HIGH){
    Serial.print(StartStop); Serial.print("␣"); Serial.println(servo);
  }

  switch (servo){// mueve los servos a la posicion correspondiente
    case 1: servo1.write(90-2-Degrees); break;//la posicin es 90 - factor de
        correccin de error de los servos - grados enviados desde el PC
    case 2: servo2.write(90-2-Degrees); break;
    case 3: servo3.write(90-2-Degrees); break;
  }

  if (StartStop == 1){//enciende o apaga el electroman
    digitalWrite(led, HIGH);
  }
  else {digitalWrite(led, LOW);}

  //leer de la camara
  blocks = pixy.getBlocks();//guarda el numero de objetos detectados en la
      bariable blocks
  if (blocks){// si se ha detectado algo
    j++;
    //como la camara envia 50 fps, para no saturar el arduino se hace que
        procese los datos una de cada 10 veces (cada 200 ms)
    if (j%10==0){
      //sacar los datos de las imagenes
      for (k=0; k<blocks; k++){//lo hace para cada uno de los objetos detectados
        signature = pixy.blocks[k].signature;//guarda en la variable signature
            la asignacin del objeto detectado (puede ir de 1 a 7)
        x = map(pixy.blocks[k].x, 0, 300, 0, 100);//coje la posicin x del objeto
             y lo pasa de un rango de 0 a 300 a un rango de 0 a 100
        y = map(pixy.blocks[k].y, 0, 200, 100, 0);//coje la posicin y del objeto
             y lo pasa de un rango de 0 a 200 a un rango de 0 a 100
        trama = s + "␣" + inicio + "␣" + signature + "␣" + x +     "␣" + y + "␣"
             + final;// construye una trama con la siguiente estructura: s
            inicio_de_trama asignacion_del_objeto posx posy final_de_trama
        //el inicio y final de trama estan puestos para evitar datos erroneos en
             la recepcion del PC
        Serial.println(trama);// envia la trama por serie al PC
      }
    }
  }
}
```

3 mainwindow.cpp

```
#include "mainwindow.h"
#include "ui_mainwindow.h"
#include <stdio.h>

char position1[1000];
char position2[1000];
char position3[1000];
char trama[1000];
char ini[1000];
char serv[1000];

int check;
/**
 * @brief MainWindow::MainWindow  nica clase de la aplicaci n. Dibuja la gui,
     se encarga de la selecci n de botones, configura todas las signals y slots
     e inicializa los temporizadores
 * @param parent
 */
MainWindow::MainWindow(QWidget *parent) :
    QMainWindow(parent),
    ui(new Ui::MainWindow)
{
    ui->setupUi(this);

    _timer = new QTimer(this);
    connect(_timer,SIGNAL(timeout()),this, SLOT(update()));
    _timer->start(50);

    _timer2 = new QTimer(this);
    connect(_timer2,SIGNAL(timeout()),this, SLOT(updates()));
    _timer2->start(25);

    _timer3 = new QTimer(this);
    connect(_timer3,SIGNAL(timeout()),this, SLOT(automode()));

    _port = new QSerialPort();
    _port->setBaudRate(QSerialPort::Baud115200);
    _port->setDataBits(QSerialPort::Data8);
    _port->setParity(QSerialPort::NoParity);
    _port->setStopBits(QSerialPort::OneStop);
    _port->setFlowControl(QSerialPort::NoFlowControl);
    connect(_port,SIGNAL(readyRead()),this,SLOT(LeerPuerto()));
}
/**
 * @brief MainWindow::~MainWindow
 */
MainWindow::~MainWindow()
{
    delete ui;
}
/**
 * @brief MainWindow::update  m todo encargado de la composici n y env o de la
      trama para control de los actuadores en arduino y actualizar la trama en la
      GUI
 */
```

```
void MainWindow::update()
{
    ui->Inibyte->setText(QString::number(_initbyte));//convierte int a Qstring

    if (counter < 3){
        counter++;
    }
    else {
        counter = 1;
    }
    ui->Servo->setText(QString::number(counter));

    ui->Pos->setText(QString::number(_pos));
    if (_pos < 0){ui->Sign->setText("1");}
    else {ui->Sign->setText("0");}
    ui->Pos_2->setText(QString::number(_pos2));
    if (_pos2 < 0){ui->Sign_2->setText("1");}
    else {ui->Sign_2->setText("0");}
    ui->Pos_3->setText(QString::number(_pos3));
    if (_pos3 < 0){ui->Sign_3->setText("1");}
    else {ui->Sign_3->setText("0");}

    if (_transmit == 1){
        sprintf(position1,"%d",_pos);
        sprintf(position2,"%d",_pos2);
        sprintf(position3,"%d",_pos3);
        sprintf(ini,"%d",_initbyte);
        sprintf(serv,"%d",counter);
        strcpy(trama,ini);
        strcat(trama, "␣");
        strcat(trama, serv);
        strcat(trama,"␣");
        switch (counter) {
        case 1:
            if (_pos >= 0){
                strcat(trama,"0␣");
            }
            else if (_pos < 0){
                strcat(trama,"1␣");
                _pos = -_pos;
            }
            if (_pos >=0 && _pos <10){
                strcat(trama,"00");
            }
            else if (_pos >= 10 && _pos<100){
                strcat(trama,"0");
            }
            sprintf(position1,"%d",_pos);
            strcat(trama, position1);
            break;
        case 2:
            if (_pos2 >= 0){
                strcat(trama,"0␣");
            }
            else if (_pos2 < 0){
                strcat(trama,"1␣");
                _pos2 = -_pos2;
```

```
            }
            if (_pos2 >=0 && _pos2 <10){
                strcat(trama,"00");
            }
            else if (_pos2 >= 10 && _pos2<100){
                strcat(trama,"0");
            }
            sprintf(position2,"%d",_pos2);
            strcat(trama, position2);
            break;
        case 3:
            if (_pos3 >= 0){
                strcat(trama,"0␣");
            }
            else if (_pos3 < 0){
                strcat(trama,"1␣");
                _pos3 = -_pos3;
            }
            if (_pos3 >=0 && _pos3 <10){
                strcat(trama,"00");
            }
            else if (_pos3 >= 10 && _pos3<100){
                strcat(trama,"0");
            }
            sprintf(position3,"%d",_pos3);
            strcat(trama, position3);
            break;
        default:
            break;
        }
        strcat(trama,"\n\r");
        _port->write(trama);
    }
}
/**
 * @brief MainWindow::on_horizontalSlider_valueChanged m todo que se encarga de
      actualizar un display del cual se toman datos en otros m todos y funciones
 * @param value valor del slider y valor a mostrar por en el display
 */
void MainWindow::on_horizontalSlider_valueChanged(int value)
{
    ui->lcdNumber_8->display(value);
}
/**
 * @brief MainWindow::on_InitStop_clicked m todo que se encarga de dar la orden
      de apagar o encender el electroim n
 */
void MainWindow::on_InitStop_clicked()
{
    ui->InitStop->setText("Stop");
    if (_initbyte == 0){
        _initbyte = 1;
        ui->InitStop->setText("Stop");
    }
    else {
        _initbyte = 0;
        ui->InitStop->setText("Init");
```

```
    }
}
/**
 * @brief MainWindow::on_transmit_clicked este m todo se encarga de intentar
     abrir el puerto y avisa en caso de exito o de fracaso
 */
void MainWindow::on_transmit_clicked()
{
    _port->setPortName("/dev/pts/11");
    printf("Abriendo␣el␣puerto␣...␣");
    if (_port->open(QIODevice::ReadWrite)) {
    printf("abierto!!!\n");
    _transmit = 1;
    }
    else {
    printf("vaya,␣esto␣CASCA\n");
    }
}
/**
 * @brief MainWindow::on_transmitStop_clicked este metodo cierra el puerto
 */
void MainWindow::on_transmitStop_clicked()
{
    _port->close();
    _transmit = 0;
}
/**
 * @brief MainWindow::on_verticalSlider_valueChanged m todo que se encarga de
     actualizar un display del cual se toman datos en otros m todos y funciones
 * @param value valor del slider y valor a mostrar por en el display
 */
void MainWindow::on_verticalSlider_valueChanged(int value)
{
    ui->lcdNumber_9->display(value);
}
/**
 * @brief MainWindow::on_verticalSlider_2_valueChanged m todo que se encarga de
      actualizar un display del cual se toman datos en otros m todos y funciones
 * @param value valor del slider y valor a mostrar por en el display
 */
void MainWindow::on_verticalSlider_2_valueChanged(int value)
{
    ui->lcdNumber_10->display(value);
}
/**
 * @brief MainWindow::delta_calcForward funci n que calcula la cinem tica
     inversa
 * @param theta1  ngulo del servo 1
 * @param theta2  ngulo del servo 2
 * @param theta3  ngulo del servo 3
 * @return este valor dice si es una posici n posible o no
 */
float MainWindow::delta_calcForward(float theta1, float theta2, float theta3)
{
    float t = (f-e)*tan30/2;
    float dtr = pi/(float)180.0;
```

```
    float result[4];

    theta1 *= dtr;
    theta2 *= dtr;
    theta3 *= dtr;

    float y1 = -(t + rf*cos(theta1));
    float z1 = -rf*sin(theta1);

    float y2 = (t + rf*cos(theta2))*sin30;
    float x2 = y2*tan60;
    float z2 = -rf*sin(theta2);

    float y3 = (t + rf*cos(theta3))*sin30;
    float x3 = -y3*tan60;
    float z3 = -rf*sin(theta3);

    float dnm = (y2-y1)*x3-(y3-y1)*x2;

    float w1 = y1*y1 + z1*z1;
    float w2 = x2*x2 + y2*y2 + z2*z2;
    float w3 = x3*x3 + y3*y3 + z3*z3;

    // x = (a1*z + b1)/dnm
    float a1 = (z2-z1)*(y3-y1)-(z3-z1)*(y2-y1);
    float b1 = -((w2-w1)*(y3-y1)-(w3-w1)*(y2-y1))/2.0;

    // y = (a2*z + b2)/dnm;
    float a2 = -(z2-z1)*x3+(z3-z1)*x2;
    float b2 = ((w2-w1)*x3 - (w3-w1)*x2)/2.0;

    // a*z^2 + b*z + c = 0
    float a = a1*a1 + a2*a2 + dnm*dnm;
    float b = 2*(a1*b1 + a2*(b2-y1*dnm) - z1*dnm*dnm);
    float c = (b2-y1*dnm)*(b2-y1*dnm) + b1*b1 + dnm*dnm*(z1*z1 - re*re);

    // discriminant
    float d = b*b - (float)4.0*a*c;
    if (d < 0) {
        return -1;
    }// non-existing point

    posZ0 = -(float)0.5*(b+sqrt(d))/a;
    posX0 = (a1*posZ0 + b1)/dnm;
    posY0 = (a2*posZ0 + b2)/dnm;

    return 0;
}
/**
 * @brief MainWindow::delta_calcAngleYZ Funci n que calcula la cinem atica
     inversa (solo uno de los  ngulos)
 * @param x0 posici n X del elector final
 * @param y0 posici n Y del efector final
 * @param z0 posici n Z del efector final
 * @return  ngulo que ha de tomar el servo
 */
float MainWindow::delta_calcAngleYZ(float x0, float y0, float z0)
```

```
{
    float y1 = -0.5 * 0.57735 * f; // f/2 * tg 30
    y0 -= 0.5 * 0.57735    * e;    // shift center to edge
    // z = a + b*y
    float a = (x0*x0 + y0*y0 + z0*z0 +rf*rf - re*re - y1*y1)/(2*z0);
    float b = (y1-y0)/z0;
    // discriminant
    float d = -(a+b*y1)*(a+b*y1)+rf*(b*b*rf+rf);
    if (d < 0) return -1; // non-existing point
    float yj = (y1 - a*b - sqrt(d))/(b*b + 1); // choosing outer point
    float zj = a + b*yj;
    float theta = 180.0*atan(-zj/(y1 - yj))/pi + ((yj>y1)?180.0:0.0);
    return theta;
}
/**
 * @brief MainWindow::updates m todo que llama las funciones de c lculo de
     cinem tica y actualiza los valores de unos displays con los resultados
 */
void MainWindow::updates()
{
    ui->lcdNumber->display(ui->lcdNumber_11->value());
    ui->lcdNumber_2->display(ui->lcdNumber_12->value());
    ui->lcdNumber_3->display(ui->lcdNumber_13->value());

    check = delta_calcForward(ui->lcdNumber->value(), ui->lcdNumber_2->value(),
        ui->lcdNumber_3->value());

    ui->lcdNumber_4->display(posX0);
    ui->lcdNumber_5->display(posY0);
    ui->lcdNumber_6->display(posZ0);
    ui->lcdNumber_7->display(check);

    theta1 = delta_calcAngleYZ(ui->lcdNumber_8->value(),
                               ui->lcdNumber_9->value(),
                               ui->lcdNumber_10->value());
    theta2 = delta_calcAngleYZ(ui->lcdNumber_8->value()*cos120 + ui->lcdNumber_9
        ->value()*sin120,
                               ui->lcdNumber_9->value()*cos120-ui->lcdNumber_8->
                                   value()*sin120,
                               ui->lcdNumber_10->value());  // rotate coords to
                                   +120 deg
    theta3 = delta_calcAngleYZ(ui->lcdNumber_8->value()*cos120 - ui->lcdNumber_9
        ->value()*sin120,
                               ui->lcdNumber_9->value()*cos120+ui->lcdNumber_8->
                                   value()*sin120,
                               ui->lcdNumber_10->value());
    ui->lcdNumber_11->display(theta1);
    ui->lcdNumber_12->display(theta2);
    ui->lcdNumber_13->display(theta3);

    _pos = ui->lcdNumber_11->value();
    _pos2 = ui->lcdNumber_12->value();
    _pos3 = ui->lcdNumber_13->value();
}
/**
 * @brief MainWindow::LeerPuerto lee la trama que viene del arduino con los
     datos de la c mara y en caso de que la trama recivida sea correcta y se
```

```
     quiera que se actualicen los datos actualiza uns displays
 */
void MainWindow::LeerPuerto()
{
        _port->read((char*)_buffer,BUFFMAX);
        sscanf(_buffer,"%*s %d %d %d %d %d",&_inicio, &_signature, &_objx, &
            _objy, &_final);

      if (_inicio == 500 && _final == 300 && _ManualCam == true){
          ui->lcdNumber_15->display(_signature);
          ui->lcdNumber_16->display(_objx);
          ui->lcdNumber_17->display(_objy);
      }
}
/**
 * @brief MainWindow::automode m todo que sirve de demostraci n
 */
void MainWindow::automode()
{
    if (_estado == 1){//arriba en inicio
        ui->horizontalSlider->setValue(56);
        ui->verticalSlider->setValue(-80);
        ui->verticalSlider_2->setValue(-120);
        _timer3->stop();
        _timer3->start(500);
        _estado = 2;//2
    }
    else if (_estado == 2){//abajo en inicio
        ui->verticalSlider_2->setValue(-168);
        _initbyte = 1;
        _estado = 3;
    }
    else if (_estado == 3){
        ui->verticalSlider_2->setValue(-120);
        _estado = 4;
    }
    else if (_estado == 4){
        ui->horizontalSlider->setValue(0);
        ui->verticalSlider->setValue(0);
        ui->lcdNumber_15->display(0);
        if (_color == 1){
            _estado = 5;//3
            _color = 0;
        }
        else if (_color == 2){
            _estado = 8;
            _color = 0;
        }
    }
    else if (_estado == 5){//arriba rojo
        ui->horizontalSlider->setValue(69);
        ui->verticalSlider->setValue(80);
        ui->verticalSlider_2->setValue(-120);
        _estado = 6;//4
    }
    else if (_estado == 6){//abajo rojo iman encendido
        ui->verticalSlider_2->setValue(-168);
```

```
        _estado = 7;
    }
    else if (_estado == 7){//abajo rojo im n apagado
        _initbyte = 0;
        ui->lcdNumber_15->display(0);
        _timer3->stop();
        _timer3->start(100);
        _estado = 0;
    }
    else if (_estado == 8){//arriba azul
        ui->horizontalSlider->setValue(-56);
        ui->verticalSlider->setValue(80);
        ui->verticalSlider_2->setValue(-120);
        _estado = 9;
    }
    else if (_estado == 9){//abajo azul iman encendido
        ui->verticalSlider_2->setValue(-168);
        _estado = 10;
    }
    else if (_estado == 10){// abajo azul im n apagado
        _initbyte = 0;
         ui->lcdNumber_15->display(0);
         _timer3->stop();
         _timer3->start(100);
        _estado = 0;
    }
    else if (_estado == 0){
        ui->horizontalSlider->setValue(0);
        ui->verticalSlider->setValue(0);
        ui->verticalSlider_2->setValue(-120);
        _color = ui->lcdNumber_15->value();
        _posX = ui->lcdNumber_16->value();
        _posY = ui-> lcdNumber_17->value();
        if (_color == 1 && _posX < 60 && _posX > 40 && _posY < 55 && _posY > 35
            ||
            _color == 2 && _posX < 60 && _posX > 40 && _posY < 55 && _posY > 45
                ){
            _estado = 1;
            _timer3->stop();
            _timer3->start(100);
        }
    }
}
/**
 * @brief MainWindow::on_InitStop_2_clicked m todo que dice si se quiere que
     los datos de la c mara sean actualizados o no
 */
void MainWindow::on_InitStop_2_clicked()
{
    if (_ManualCam == false){
        _ManualCam = true;
        ui->InitStop_2->setText("Manual");
    }
    else {
        _ManualCam = false;
        ui->InitStop_2->setText("Camera");
    }
```

```
}
/**
 * @brief MainWindow::on_pushButton_clicked m todo que activa o desactiva la
     demostraci n
 */
void MainWindow::on_pushButton_clicked()
{
    if (_auto == 0){
        ui->pushButton->setText("Stop␣Auto");
        _auto = 1;
        _timer3->start(100);
        _estado = 0;
    }
    else {
        ui->pushButton->setText("Start␣Auto");
        _auto = 0;
        _timer3->stop();
        _estado = 20;
    }
}
```

4 mainwindow.h

```
#ifndef MAINWINDOW_H
#define MAINWINDOW_H

#include <QMainWindow>
#include <QTimer>
#include <QSerialPort>
#include <stdio.h>
#include <math.h>

#define BUFFMAX 1000

namespace Ui {
class MainWindow;
}

class MainWindow : public QMainWindow
{
    Q_OBJECT

public:
    explicit MainWindow(QWidget *parent = 0);
    ~MainWindow();
    float delta_calcForward(float theta1, float theta2, float theta3);
    float delta_calcAngleYZ(float x0, float y0, float z0);

public slots:
    void update (void);

    void updates(void);

    void LeerPuerto(void);

    void automode(void);

private slots:
    void on_horizontalSlider_valueChanged(int value);

    void on_InitStop_clicked();

    void on_transmit_clicked();

    void on_transmitStop_clicked();

    void on_verticalSlider_valueChanged(int value);

    void on_verticalSlider_2_valueChanged(int value);

    void on_InitStop_2_clicked();

    void on_pushButton_clicked();

private:
    Ui::MainWindow *ui;
    QTimer *_timer;
    QTimer *_timer2;
```

```
    QTimer *_timer3;

    int _initbyte=0;
    int _servo;
    int _pos=90;
    int _pos2=90;
    int _pos3=90;
    char _posi;
    QSerialPort *_port;
    int _transmit=0;
    int counter = 1;
    int _objx;
    int _objy;
    int _signature;
    int _inicio;
    int _final;
    char _buffer[BUFFMAX];
    bool _ManualCam = false;

    // robot geometry
    // (look at pics above for explanation)
    const float e = 64;//115.0;     // end effector
    const float f = 220;//457.3;     // base
    const float re = 175;//232.0;
    const float rf = 50;//112.0;

    // trigonometric constants
    const float sqrt3 = sqrt(3.0);
    const float pi = 3.141592653;    // PI
    const float sin120 = sqrt3/2.0;
    const float cos120 = -0.5;
    const float tan60 = sqrt3;
    const float sin30 = 0.5;
    const float tan30 = 1/sqrt3;

    float theta1 = 10;
    float theta2 = 10;
    float theta3 = 10;

    float posX0 = 10;
    float posY0 = 10;
    float posZ0 = 10;

    //auto mode
    int _auto = 0;
    int _color = 0;
    int _estado = 0;
    int _posX;
    int _posY;
};

#endif // MAINWINDOW_H
```

5 mainwindow.ui

```
<?xml version="1.0" encoding="UTF-8"?>
<ui version="4.0">
 <class>MainWindow</class>
 <widget class="QMainWindow" name="MainWindow">
  <property name="geometry">
   <rect>
    <x>0</x>
    <y>0</y>
    <width>1143</width>
    <height>830</height>
   </rect>
  </property>
  <property name="windowTitle">
   <string>MainWindow</string>
  </property>
  <widget class="QWidget" name="centralWidget">
   <widget class="QPushButton" name="InitStop">
    <property name="geometry">
     <rect>
      <x>50</x>
      <y>20</y>
      <width>101</width>
      <height>23</height>
     </rect>
    </property>
    <property name="text">
     <string>Init</string>
    </property>
   </widget>
   <widget class="QLabel" name="label">
    <property name="geometry">
     <rect>
      <x>110</x>
      <y>210</y>
      <width>81</width>
      <height>41</height>
     </rect>
    </property>
    <property name="font">
     <font>
      <pointsize>18</pointsize>
     </font>
    </property>
    <property name="text">
     <string>Trama</string>
    </property>
   </widget>
   <widget class="QLabel" name="Inibyte">
    <property name="geometry">
     <rect>
      <x>210</x>
      <y>210</y>
      <width>81</width>
      <height>41</height>
     </rect>
```

```
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLabel" name="Servo">
 <property name="geometry">
  <rect>
   <x>300</x>
   <y>210</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLabel" name="Pos">
 <property name="geometry">
  <rect>
   <x>480</x>
   <y>210</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QPushButton" name="transmit">
 <property name="geometry">
  <rect>
   <x>270</x>
   <y>20</y>
   <width>101</width>
   <height>23</height>
  </rect>
 </property>
 <property name="text">
  <string>Transmit</string>
```

```
 </property>
</widget>
<widget class="QPushButton" name="transmitStop">
 <property name="geometry">
  <rect>
   <x>370</x>
   <y>20</y>
   <width>101</width>
   <height>23</height>
  </rect>
 </property>
 <property name="text">
  <string>Stop</string>
 </property>
</widget>
<widget class="QLabel" name="Pos_2">
 <property name="geometry">
  <rect>
   <x>480</x>
   <y>240</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLabel" name="Pos_3">
 <property name="geometry">
  <rect>
   <x>480</x>
   <y>270</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLCDNumber" name="lcdNumber_7">
 <property name="geometry">
  <rect>
   <x>470</x>
   <y>380</y>
   <width>118</width>
```

```
    <height>41</height>
   </rect>
  </property>
 </widget>
 <widget class="QWidget" name="verticalLayoutWidget_2">
  <property name="geometry">
   <rect>
    <x>90</x>
    <y>310</y>
    <width>371</width>
    <height>191</height>
   </rect>
  </property>
  <layout class="QVBoxLayout" name="verticalLayout_2">
   <item>
    <layout class="QGridLayout" name="gridLayout">
     <item row="0" column="2">
      <widget class="QLabel" name="label_2">
       <property name="text">
        <string>Theta 3</string>
       </property>
      </widget>
     </item>
     <item row="0" column="1">
      <widget class="QLabel" name="label_3">
       <property name="text">
        <string>Theta 2</string>
       </property>
      </widget>
     </item>
     <item row="1" column="0">
      <widget class="QLCDNumber" name="lcdNumber">
       <property name="segmentStyle">
        <enum>QLCDNumber::Flat</enum>
       </property>
       <property name="value" stdset="0">
        <double>40.000000000000000</double>
       </property>
      </widget>
     </item>
     <item row="1" column="1">
      <widget class="QLCDNumber" name="lcdNumber_2">
       <property name="value" stdset="0">
        <double>50.000000000000000</double>
       </property>
      </widget>
     </item>
     <item row="1" column="2">
      <widget class="QLCDNumber" name="lcdNumber_3">
       <property name="value" stdset="0">
        <double>60.000000000000000</double>
       </property>
      </widget>
     </item>
     <item row="0" column="0">
      <widget class="QLabel" name="label_4">
       <property name="text">
```

```
       <string>Theta 1</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QGridLayout" name="gridLayout_2">
    <item row="0" column="2">
     <widget class="QLabel" name="label_7">
      <property name="text">
       <string>Z0</string>
      </property>
     </widget>
    </item>
    <item row="0" column="0">
     <widget class="QLabel" name="label_8">
      <property name="text">
       <string>X0</string>
      </property>
     </widget>
    </item>
    <item row="0" column="1">
     <widget class="QLabel" name="label_10">
      <property name="text">
       <string>Y0</string>
      </property>
     </widget>
    </item>
    <item row="1" column="0">
     <widget class="QLCDNumber" name="lcdNumber_4"/>
    </item>
    <item row="1" column="1">
     <widget class="QLCDNumber" name="lcdNumber_5"/>
    </item>
    <item row="1" column="2">
     <widget class="QLCDNumber" name="lcdNumber_6"/>
    </item>
   </layout>
  </item>
 </layout>
</widget>
<widget class="QLabel" name="label_13">
 <property name="geometry">
  <rect>
   <x>760</x>
   <y>520</y>
   <width>57</width>
   <height>15</height>
  </rect>
 </property>
 <property name="text">
  <string>Inverse</string>
 </property>
</widget>
<widget class="QLCDNumber" name="lcdNumber_14">
 <property name="geometry">
```

```
  <rect>
   <x>990</x>
   <y>380</y>
   <width>118</width>
   <height>41</height>
  </rect>
 </property>
</widget>
<widget class="QWidget" name="verticalLayoutWidget_3">
 <property name="geometry">
  <rect>
   <x>610</x>
   <y>310</y>
   <width>371</width>
   <height>191</height>
  </rect>
 </property>
 <layout class="QVBoxLayout" name="verticalLayout_3">
  <item>
   <layout class="QGridLayout" name="gridLayout_3">
    <item row="0" column="2">
     <widget class="QLabel" name="label_5">
      <property name="text">
       <string>Z0</string>
      </property>
     </widget>
    </item>
    <item row="0" column="1">
     <widget class="QLabel" name="label_6">
      <property name="text">
       <string>Y0</string>
      </property>
     </widget>
    </item>
    <item row="1" column="1">
     <widget class="QLCDNumber" name="lcdNumber_9">
      <property name="value" stdset="0">
       <double>0.000000000000000</double>
      </property>
     </widget>
    </item>
    <item row="1" column="2">
     <widget class="QLCDNumber" name="lcdNumber_10">
      <property name="value" stdset="0">
       <double>-170.000000000000000</double>
      </property>
      <property name="intValue" stdset="0">
       <number>-170</number>
      </property>
     </widget>
    </item>
    <item row="0" column="0">
     <widget class="QLabel" name="label_9">
      <property name="text">
       <string>X0</string>
      </property>
     </widget>
```

```
    </item>
    <item row="1" column="0">
     <widget class="QLCDNumber" name="lcdNumber_8">
      <property name="value" stdset="0">
       <double>0.000000000000000</double>
      </property>
     </widget>
    </item>
   </layout>
  </item>
  <item>
   <layout class="QGridLayout" name="gridLayout_4">
    <item row="0" column="2">
     <widget class="QLabel" name="label_11">
      <property name="text">
       <string>theta 3</string>
      </property>
     </widget>
    </item>
    <item row="0" column="1">
     <widget class="QLabel" name="label_15">
      <property name="text">
       <string>Theta 2</string>
      </property>
     </widget>
    </item>
    <item row="1" column="0">
     <widget class="QLCDNumber" name="lcdNumber_11"/>
    </item>
    <item row="1" column="1">
     <widget class="QLCDNumber" name="lcdNumber_12"/>
    </item>
    <item row="1" column="2">
     <widget class="QLCDNumber" name="lcdNumber_13"/>
    </item>
    <item row="0" column="0">
     <widget class="QLabel" name="label_12">
      <property name="text">
       <string>theta 1</string>
      </property>
     </widget>
    </item>
   </layout>
  </item>
 </layout>
</widget>
<widget class="QLabel" name="label_16">
 <property name="geometry">
  <rect>
   <x>250</x>
   <y>520</y>
   <width>57</width>
   <height>15</height>
  </rect>
 </property>
 <property name="text">
  <string>Direct</string>
```

```
 </property>
</widget>
<widget class="QWidget" name="gridLayoutWidget">
 <property name="geometry">
  <rect>
   <x>610</x>
   <y>10</y>
   <width>261</width>
   <height>261</height>
  </rect>
 </property>
 <layout class="QGridLayout" name="gridLayout_5">
  <item row="0" column="0">
   <widget class="QSlider" name="horizontalSlider">
    <property name="minimum">
     <number>-80</number>
    </property>
    <property name="maximum">
     <number>80</number>
    </property>
    <property name="value">
     <number>0</number>
    </property>
    <property name="orientation">
     <enum>Qt::Horizontal</enum>
    </property>
   </widget>
  </item>
  <item row="0" column="1">
   <widget class="QSlider" name="verticalSlider">
    <property name="toolTipDuration">
     <number>-4</number>
    </property>
    <property name="minimum">
     <number>-80</number>
    </property>
    <property name="maximum">
     <number>80</number>
    </property>
    <property name="orientation">
     <enum>Qt::Vertical</enum>
    </property>
   </widget>
  </item>
  <item row="0" column="2">
   <widget class="QSlider" name="verticalSlider_2">
    <property name="minimum">
     <number>-220</number>
    </property>
    <property name="maximum">
     <number>-120</number>
    </property>
    <property name="value">
     <number>-170</number>
    </property>
    <property name="orientation">
     <enum>Qt::Vertical</enum>
```

```
    </property>
   </widget>
  </item>
 </layout>
</widget>
<widget class="QLabel" name="label_17">
 <property name="geometry">
  <rect>
   <x>690</x>
   <y>280</y>
   <width>57</width>
   <height>15</height>
  </rect>
 </property>
 <property name="text">
  <string>X</string>
 </property>
</widget>
<widget class="QLabel" name="label_18">
 <property name="geometry">
  <rect>
   <x>830</x>
   <y>280</y>
   <width>57</width>
   <height>15</height>
  </rect>
 </property>
 <property name="text">
  <string>Y</string>
 </property>
</widget>
<widget class="QLabel" name="label_19">
 <property name="geometry">
  <rect>
   <x>860</x>
   <y>280</y>
   <width>57</width>
   <height>15</height>
  </rect>
 </property>
 <property name="text">
  <string>Z</string>
 </property>
</widget>
<widget class="QLabel" name="Sign">
 <property name="geometry">
  <rect>
   <x>380</x>
   <y>210</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
```

```
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLabel" name="Sign_2">
 <property name="geometry">
  <rect>
   <x>380</x>
   <y>240</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLabel" name="Sign_3">
 <property name="geometry">
  <rect>
   <x>380</x>
   <y>270</y>
   <width>81</width>
   <height>41</height>
  </rect>
 </property>
 <property name="font">
  <font>
   <pointsize>18</pointsize>
  </font>
 </property>
 <property name="text">
  <string>Trama</string>
 </property>
</widget>
<widget class="QLCDNumber" name="lcdNumber_15">
 <property name="geometry">
  <rect>
   <x>100</x>
   <y>100</y>
   <width>101</width>
   <height>51</height>
  </rect>
 </property>
</widget>
<widget class="QLCDNumber" name="lcdNumber_16">
 <property name="geometry">
  <rect>
   <x>220</x>
   <y>100</y>
   <width>101</width>
```

```
    <height>51</height>
   </rect>
  </property>
 </widget>
 <widget class="QLCDNumber" name="lcdNumber_17">
  <property name="geometry">
   <rect>
    <x>340</x>
    <y>100</y>
    <width>101</width>
    <height>51</height>
   </rect>
  </property>
 </widget>
 <widget class="QPushButton" name="InitStop_2">
  <property name="geometry">
   <rect>
    <x>160</x>
    <y>20</y>
    <width>101</width>
    <height>23</height>
   </rect>
  </property>
  <property name="text">
   <string>Camera</string>
  </property>
 </widget>
 <widget class="QPushButton" name="pushButton">
  <property name="geometry">
   <rect>
    <x>470</x>
    <y>20</y>
    <width>80</width>
    <height>23</height>
   </rect>
  </property>
  <property name="text">
   <string>Start Auto</string>
  </property>
 </widget>
</widget>
<widget class="QMenuBar" name="menuBar">
 <property name="geometry">
  <rect>
   <x>0</x>
   <y>0</y>
   <width>1143</width>
   <height>20</height>
  </rect>
 </property>
</widget>
<widget class="QToolBar" name="mainToolBar">
 <attribute name="toolBarArea">
  <enum>TopToolBarArea</enum>
 </attribute>
 <attribute name="toolBarBreak">
  <bool>false</bool>
```

```
   </attribute>
  </widget>
  <widget class="QStatusBar" name="statusBar"/>
 </widget>
 <layoutdefault spacing="6" margin="11"/>
 <resources/>
 <connections/>
</ui>
```