Universidad Autónoma Metropolitana
Unidad Azcapotzalco
División de Ciencias Básicas e Ingeniería

Licenciatura en Ingeniería en Computación

Proyecto Tecnológico

Programación del robot Robotis Darwin-OP con python

Luis Alberto Leyva Hernández

210200159

Trimestre 2014 Primavera

16 de julio de 2014

M. en C. José Alfredo Estrada Soto

Profesor Titular ”C”

Departamento de Electrónica

Yo, M. en C. José Alfredo Estrada Soto, declaro que aprobé el contenido del presente Reporte de Proyecto de Integración y doy mi autorización para su publicación en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma del asesor

Yo, Luis Alberto Leyva Hernández, doy mi autorización a la Coordinación de Servicios de Información de la Universidad Autónoma Metropolitana, Unidad Azcapotzalco, para publicar el presente documento en la Biblioteca Digital, así como en el Repositorio Institucional de UAM Azcapotzalco.

Firma del alumno

”No tengo que ’tener’ una respuesta.
No me siento aterrorizado por no conocer cosas,
por estar perdido en el misterioso universo sin tener ningún propósito;
que es el modo en el que la realidad es,
hasta donde puedo decir, posiblemente.
Esto no me aterra. ”
Richard Feynman

”La Ciencia es lo que entendemos lo suficientemente bien
como para explicárselo a una computadora;
todo lo demás que hacemos es Arte.”
Donald Knuth

Resumen

Hoy en día la robótica es de suma importancia en la vida diaria, los robots son máquinas diseñadas y fabricadas para realizar distintas tareas que benefician a la sociedad. Los robots pueden realizar las tareas que les son asignadas de forma automatizada o guiados por un operador humano. Ambas formas son logradas mediante el uso de programas que controlen y operen al robot.

El desarrollo de programas puede volverse una labor compleja si la API proporcionada para programarlos requiere un conocimiento profundo de las bibliotecas que incluye, es decir si hace falta conocer a detalle un conjunto amplio de parámetros sobre los componentes y movimientos del robot, lo que implica un mayor gasto de tiempo en el estudio y aprendizaje de la API que en la programación del robot. Una programación compleja, además de consumir más tiempo, es más propensa a errores al implementar los algoritmos que controlan a un robot.

El departamento de electrónica de la UAM-Azcapotzalco cuenta con un robot Robotis Darwin-OP, en este trabajo se diseñó una API en el lenguaje de programación python que simplifica la programación de este robot. De esta manera se puede manipular al robot en una gran variedad de formas, en un lenguaje de más alto nivel.

Índice

Lista de figuras

1. Introducción

El desarrollo de robots es una actividad en constante evolución que requiere la utilización del conocimiento de distintas disciplinas como son: la electrónica, la mecánica, la física y la computación. La investigación en robótica responde a la necesidad de construir robots con aplicación en entornos reales que repercutan en beneficio de la sociedad. Actualmente la robótica se aplica exitosamente en la automatización de procesos productivos, la medicina y la exploración espacial, por mencionar algunos campos de aplicación.

La programación de un robot es una tarea fundamental en el diseño y construcción de sistemas robóticos y ambientes automatizados para lograr la adecuada implementación de algoritmos de control, inteligencia artificial y visión, entre otros. Como apoyo para escribir los programas que se encargarán de controlar la funcionalidad del robot, al desarrollar el robot suele crearse también una API[1] (del inglés Application Programming Interface) que incluya las bibliotecas necesarias para operar los componentes del robot y programar su funcionamiento (movimiento, visión, sensores).

El desarrollo de programas puede volverse una labor compleja si la API proporcionada para programarlos requiere un conocimiento profundo de las bibliotecas que incluye, es decir si hace falta conocer a detalle un conjunto amplio de parámetros sobre los componentes y movimientos del robot, lo que implica un mayor gasto de tiempo en el estudio y aprendizaje de la API que en la programación del robot. Una programación compleja, además de consumir más tiempo, es más propensa a errores al implementar los algoritmos que controlan a un robot.

Darwin-OP (Dynamic Anthromorphic Robot with Intelligence - Open Platform) es un robot humanoide miniatura de plataforma abierta desarrollado por RoMeLa en Virginia Tech con la colaboración de la Universidad de Pennsylvania, la Universidad Purdue y Robotis Co. Al ser un robot de plataforma abierta, permite la modificación tanto del software como del hardware, su objetivo es servir como herramienta en la realización de proyectos de robótica con fines educativos y de investigación. Entre sus características se encuentra poder computacional avanzado, sensores sofisticados y movimientos dinámicos.

[1] La funcionalidad básica del robot y la API proporcionada son diseñadas por el fabricante del robot.

El departamento de electrónica de la UAM-Azcapotzalco cuenta con un robot Robotis Darwin-OP[2], que es programado mediante el lenguaje de programación C++. El presente trabajo tiene como objetivo diseñar e implementar una API en el lenguaje de programación python que simplifique la programación de este robot. Facilitando la realización de proyectos con el robot.

Figura 1: Robot Robotis Darwin-OP.

[2]Sitio web del fabricante `http://www.robotis.com/xe/darwin_en`

2. Justificación

El robot Robotis Darwin-OP es programado mediante la API del fabricante, usando el lenguaje C++. El uso de esta API requiere mayor experiencia al programar, ya que permite el acceso a alto y a bajo nivel del hardware del robot, por lo cual se pueden cometer errores que incapaciten la operabilidad y funcionamiento del robot. Los métodos que incluye esta API requieren el paso de múltiples parámetros que el programador debe conocer a detalle. En programas con gran número de lineas se requiere mucho tiempo de compilación.

Simplificar la labor de programar un robot tiene como consecuencia facilitar el desarrollo de proyectos y escenarios de prueba con fines didácticos y de investigación. Python es un lenguaje de programación de alto nivel de propósito general, que resulta muy fácil de aprender. Es un lenguaje interpretado, dinámico y multiparadigma. El desarrollo de una API en python para controlar el robot, además de simplificar el proceso de programación, permite adaptar los programas que lo controlan de forma interactiva, logrando que se pueda probar o introducir una nueva característica observando los resultados inmediatamente, sin necesidad de compilar el código por cada nuevo cambio.

Así, se pretende diseñar e implementar una API en el lenguaje de programación python que simplifique la programación de este robot.

3. Antecedentes

La robótica es una rama de la ciencia y la tecnología que se encarga del estudio, diseño, construcción, operación y aplicación de robots, así como de los sistemas computacionales para su control, análisis y procesamiento de la información que reciben del entorno. Un robot es una entidad virtual o mecánica artificial. En la práctica, esto es por lo general un sistema electromecánico que, por su apariencia o sus movimientos, ofrece la sensación de tener un propósito propio. La independencia creada en sus movimientos hace que sus acciones sean la razón de un estudio profundo en el área de la ciencia y tecnología. La palabra robot puede referirse tanto a mecanismos físicos como a sistemas virtuales de software, aunque suele aludirse a los segundos con el término de bots.

Actualmente podría considerarse que un robot es una computadora con la capacidad y el propósito de movimiento que en general es capaz de desarrollar múltiples tareas de manera flexible según su programación; así que podría diferenciarse de algún electrodoméstico específico.

El concepto de crear máquinas que puedan operar de forma autónoma data desde la antigüedad, pero las máquinas totalmente autónomas no aparecieron hasta el siglo XX. El primer robot programable y dirigido de forma digital, el robot industrial Unimate, fue instalado en 1961 para levantar piezas calientes de metal de una máquina de tinte y colocarlas.

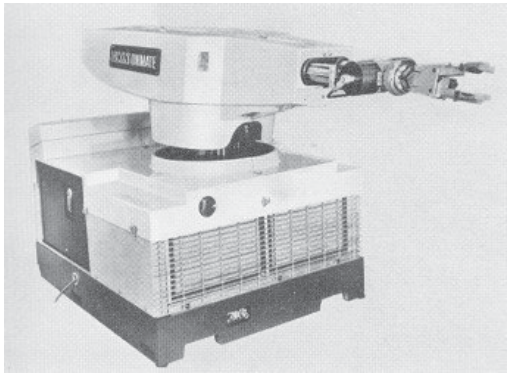

Figura 2: Robot industrial Unimate.

3.1. Los robots en la antigüedad

Una de las primeras descripciones de un autómata aparece en el texto de Lie Zi, se describe que en el primer encuentro entre el rey Mu de Zhou (1023–957 a.C.) y un ingeniero mecánico conocido como Yan Shi. Este último supuestamente presentó ante el rey su obra mecánica de forma y medidas humanas.

En el siglo IV a.C., el matemático griego Arquitas de Tarento construyó un ave mecánica que funcionaba con vapor y a la que llamó ”La paloma”. También el ingeniero Herón de Alejandría (10-70 d.C.) creó numerosos dispositivos automáticos que los usuarios podían modificar, y describió máquinas accionadas por presión de aire, vapor y agua. Por su parte, el estudioso chino Su Song levantó una torre de reloj en 1088 con figuras mecánicas que daban las campanadas al marcar las horas.

Al Jazarií[3] (1136–1206), diseñó y construyó una serie de máquinas automatizadas, entre los que había útiles de cocina, autómatas musicales que funcionaban con agua, y en 1206 los primeros robots humanoides programables. Las máquinas tenían el aspecto de cuatro músicos a bordo de un bote en un lago, entreteniendo a los invitados en las fiestas reales. Su mecanismo tenía un tambor programable con clavijas que chocaban con pequeñas palancas que accionaban instrumentos de percusión. Podían cambiarse los ritmos y patrones que tocaba el tamborilero moviendo las clavijas.

3.2. Los robots modernos

El artesano japonés Hisashige Tanaka (1799–1881), conocido como el”Edison japonés”, creó una serie de juguetes mecánicos extremadamente complejos, algunos de los cuales servían té, disparaban flechas retiradas de un carcaj[4] e incluso trazaban un kanji[5].

Por otra parte, desde la generalización del uso de la tecnología en procesos de producción con la Revolución Industrial se intentó la construcción de dispositivos automáticos que ayudaran o sustituyeran al hombre. Entre ellos destacaron los Jaquemarts, muñecos de dos o más posiciones que golpean campanas accionados por mecanismos de relojería china y japonesa.

Robots equipados con una sola rueda fueron utilizados para llevar a cabo investigaciones sobre conducta, navegación y planeo de ruta. Cuando estuvieron listos para intentar nuevamente con los robots caminantes, comenzaron con pequeños hexápodos y otros tipos de robots de múltiples patas. Estos robots imitaban insectos y artrópodos en funciones y forma. Como se ha hecho notar anteriormente, la tendencia se dirige hacia ese tipo de cuerpos que ofrecen gran flexibilidad y han probado adaptabilidad a cualquier ambiente. Con más de 4 piernas, estos robots son estáticamente estables, lo que hace que el trabajar con ellos sea más sencillo. Sólo recientemente se han hecho progresos hacia los robots con locomoción bípeda.

En 2002 Honda y Sony, comenzaron a vender comercialmente robots humanoides como ”mascotas”. Los robots con forma de perro o de serpiente se encuentran, sin embargo, en una fase de producción muy amplia, el ejemplo más notorio ha sido Aibo de Sony.

[3]Inventor musulmán de la dinastía Artuqid.

[4]Es un cilindro de piel, madera y/o tela usada por los arqueros para transportar las flechas.

[5]Caracteres utilizados en la escritura japonesa.

3.3. La robótica en la actualidad

En la actualidad, los robots comerciales e industriales son ampliamente utilizados y realizan tareas de forma más exacta o más barata que los humanos. También se les utiliza en trabajos demasiado sucios, peligrosos o tediosos para los humanos. Los robots son muy utilizados en plantas de manufactura, montaje y embalaje, en transporte, en exploraciones en la Tierra y en el espacio, cirugía, armamento, investigación en laboratorios y en la producción en masa de bienes industriales o de consumo.

Otras aplicaciones incluyen la limpieza de residuos tóxicos, minería, búsqueda y rescate de personas y localización de minas terrestres.

Los robots parecen estar abaratándose y reduciendo su tamaño, una tendencia relacionada con la miniaturización de los componentes electrónicos que se utilizan para controlarlos. Además, muchos robots son diseñados en simuladores mucho antes de construirse y de que interactúen con ambientes físicos reales.

Además de los campos mencionados, hay modelos trabajando en el sector educativo y de servicios (por ejemplo, en lugar de recepcionistas humanos o vigilancia).

4. Objetivos

4.1. Objetivo general

Desarrollar una API en el lenguaje de programación python para el desarrollo de escenarios de prueba y control de un robot humanoide.

4.2. Objetivos específicos

1. Diseñar e implementar una clase que permita acceso y control a alto nivel de los componentes del robot mediante un objeto controlador.
2. Diseñar e implementar un conjunto de clases y sus métodos que permitan programar el funcionamiento del robot.
3. Diseñar e implementar escenarios de prueba que hagan uso de las clases implementadas.

5. Marco teórico

Hay muchos tipos de robots no virtuales; son utilizados con distintos fines y en diferentes entornos. A pesar de la diferencia de sus formas y la diversidad de sus aplicaciones, todos estos robots comparten 3 similitudes básicas en su construcción.

1. Todos estos robots están construidos de forma mecánica. Usualmente la forma, tamaño y piezas mecánicas del robot son seleccionadas en función al problema que se busca resolver o a las tareas asignadas al robot para permitirle un desempeño adecuado según las condiciones físicas del entorno.

2. Cuentan con componentes eléctricos como sensores, motores, circuitos, baterías, etc. Estos componentes se encargan de recibir información desde el entorno o enviarla al mismo. También se ocupan de la operación básica del robot, ya que sin electricidad y sin motores el robot no podría moverse

3. Permiten cierto nivel de programación. Un programa puede ser entendido como el medio a través del cual un robot decide qué hacer y cuándo y cómo actuar. Incluso en un robot no automatizado se requiere un programa que permita a un usuario operador guiar al robot. Sin un programa un robot no podría realizar labores complejas.

5.1. Robot humanoide

Un robot humanoide es un robot cuyo cuerpo es construido a semejanza del cuerpo humano. Un diseño humanoide podría ser utilizado para fines funcionales: tales como la interacción con herramientas y entornos humanos; con fines experimentales: como el estudio de la locomoción bípeda, o para otros fines. En general, los robots humanoides tienen un torso, una cabeza, dos brazos y dos piernas, aunque algunas formas de robots humanoides pueden modelar sólo una parte del cuerpo, por ejemplo, de la cintura para arriba. Algunos robots humanoides pueden tener cabezas diseñado para replicar los rasgos faciales humanos, tales como los ojos y la boca. Los androides son robots humanoides construidos para parecer humanos estéticamente.

Los robots humanoides se utilizan como una herramienta de investigación en diversas áreas científicas. Los investigadores necesitan entender la estructura del cuerpo humano y su comportamiento biomecánica para construir y estudiar los robots humanoides. Por otro lado, el intento de la simulación del cuerpo humano conduce a una mejor comprensión de la misma.

La cognición humana es un campo de estudio que se centra en cómo los seres humanos aprenden de la información sensorial con el fin de adquirir las habilidades perceptivas y motoras. Este

conocimiento se utiliza para desarrollar modelos computacionales de la conducta humana y se ha ido mejorando con el tiempo.

Además de la investigación, los robots humanoides se están desarrollando para realizar tareas humanas como la asistencia personal, deben ser capaces de asistir a los enfermos y ancianos. Trabajos regulares como recepcionista o trabajador de línea de fabricación de automóviles también son adecuados para los humanoides. En esencia, ya que pueden utilizar herramientas y operar los equipos y el material diseñado para la forma humana, los humanoides podrían realizar en teoría cualquier tarea que un ser humano puede, siempre y cuando tengan el software adecuado. Sin embargo, la complejidad de hacerlo es aparentemente grande.

5.2. Darwin-OP

DARWIN-OP (Dynamic Anthropomorphic Robot with Intelligence–Open Platform) es una robot humanoide miniatura con poder computacional avanzado, sensores sofisticados, alta capacidad de carga y capacidad de movimiento dinámico desarrollado. Construido por el fabricante coreano ROBOTIS, en colaboración con el Tecnológico de Virginia, la Universidad de Purdue y la Universidad de Pennsylvania. DARWIN-OP tiene veinte grados de libertad cada uno controlado por un servomotor Dynamixel MX-28T. El MX-28T tiene un par de bloqueo de 24 kgf·cm (a 12 V, 1,5 A) y un rango de 360 gados de movimiento.

Figura 3: Diagrama de Darwin-OP.

5.2.1. Especificaciones

- Altura: 454.5 mm (17.89 pulgadas)
- Peso: 2.9 kg (6.4 libras)
- Por defecto la velocidad al caminar: 24.0 cm/s (9.5 in/s) 0.25 s/paso
- Por defecto el tiempo para levantarse desde el suelo: 2.8 s (boca abajo) y 3.9 s (boca arriba)
- Built-in PC: 1.6 GHz Intel Atom Z530 (32 bits) on-board SSD flash 4 GB
- Administrador de control (CM-730): ARM CortexM3 STM32F103RE 72 MHz
- 20 actuadores MX-28T (6 pierna DOF $\times$ 2 + 3 brazo DOF $\times$ 2 + 2 DOF cuello) con engranajes metálicos
- 3 Mbit/s bus Dynamixel de alta velocidad para el control conjunto
- Giroscopio de 3 ejes, acelerómetro de 3 ejes, botón $\times$ 3, micrófono de detección $\times$ 2
- Funcionalidad versátil (puede aceptar futuros periféricos)

Figura 4: Diagrama de servomotor.

5.2.2. Framework

Diagrama de clases del framework de Darwin

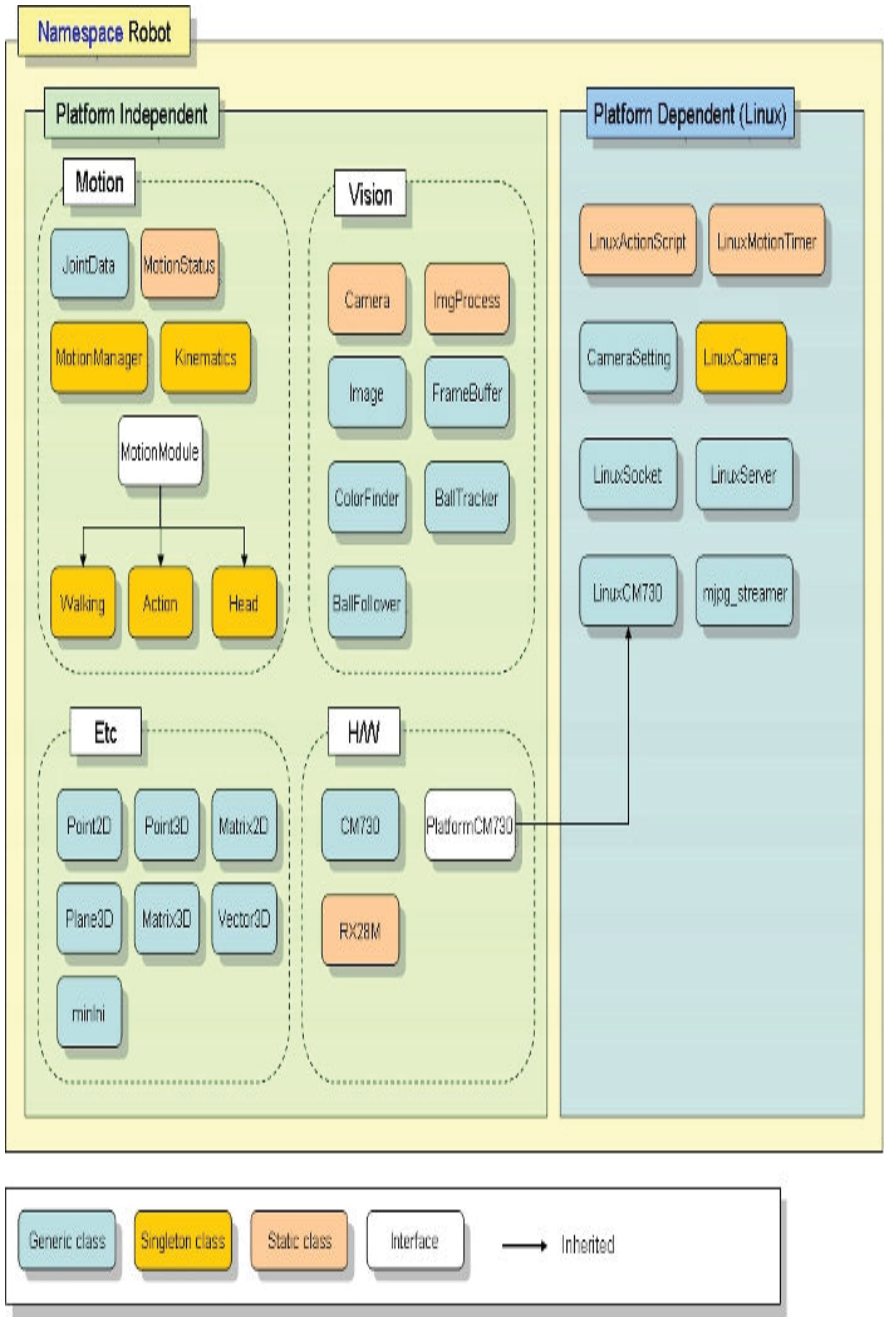

Figura 5: Diagrama de clases.

Pipeline de framework de Darwin

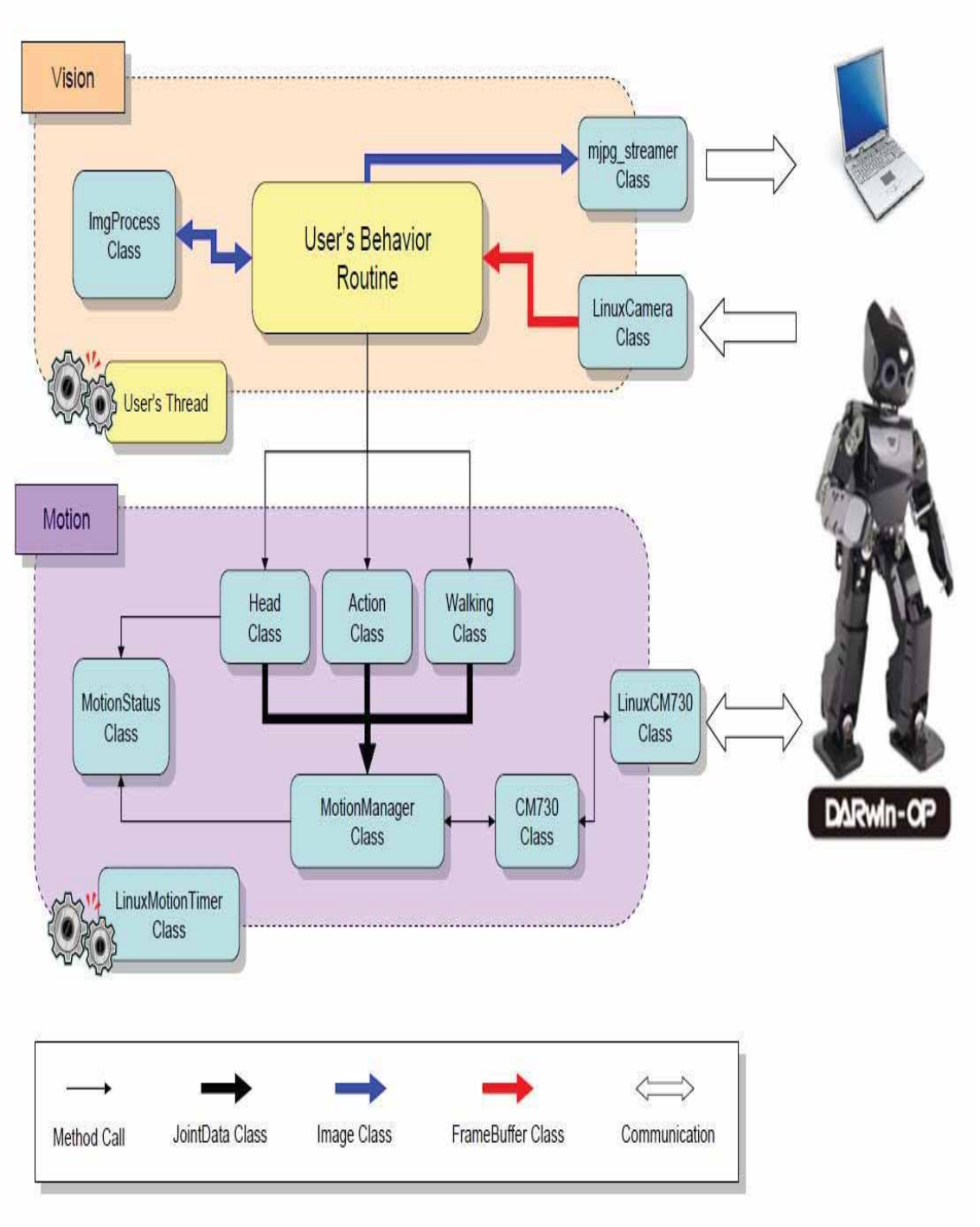

Figura 6: Diagrama de pipeline de framework.

5.3. El lenguaje de programación python

Python es un lenguaje de alto nivel multiparadigma ampliamente usado. Su filosofía de diseño enfatiza la legibilidad del código y su sintaxis permite a los programadores expresar conceptos con menos líneas de código en comparación a otros lenguajes como C. Es un lenguaje de programación fácil de aprender que cuenta con una gran biblioteca de funciones. Es interpretado y multiplataforma. Posee manejo dinámico de la memoria.

6. Desarrollo del proyecto

El trabajo desarrollado en este proyecto fue dividido en los módulos de visión, movimiento y control que se muestran en la figura 7.

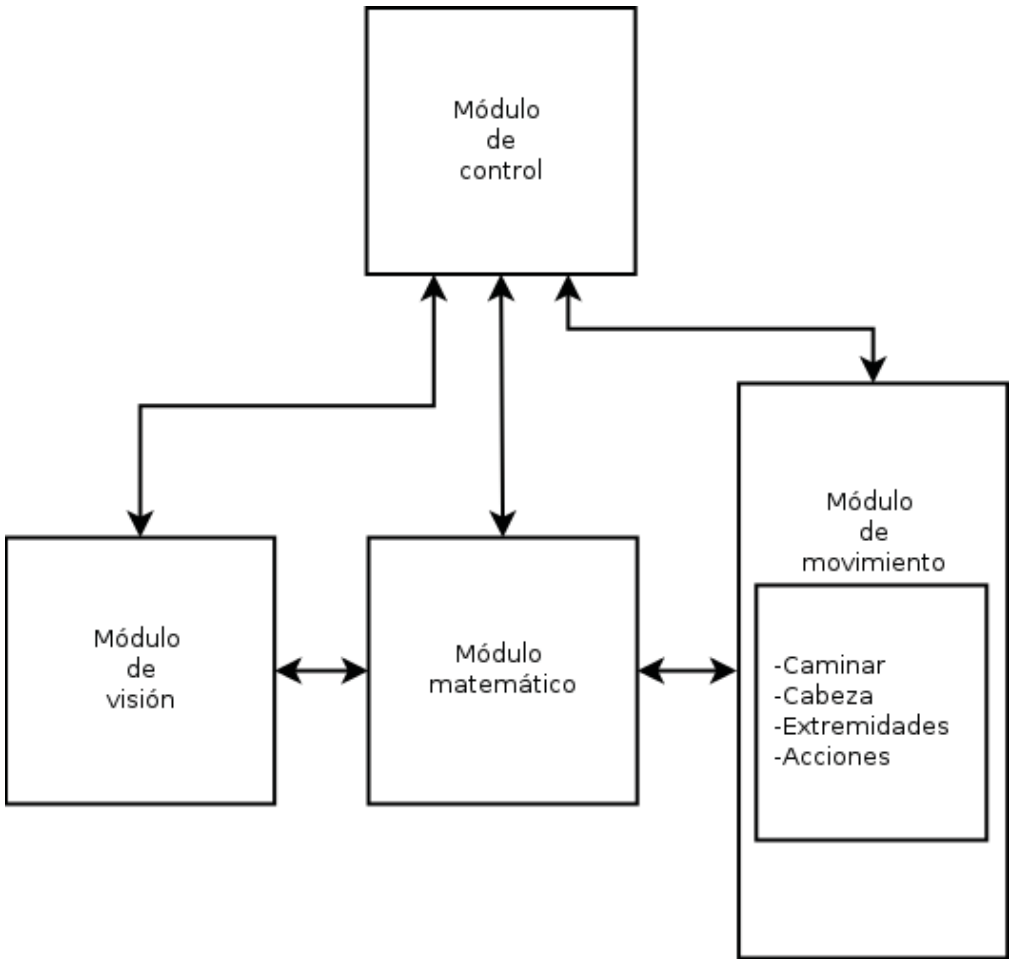

Figura 7: Módulos del proyecto.

6.1. Diagrama de casos de uso

El desarrollo del proyecto se realizó con base en los siguientes casos de uso. Debido a que toda la interacción del usuario se realiza importando el módulo de visión y el módulo de movimiento para su uso.

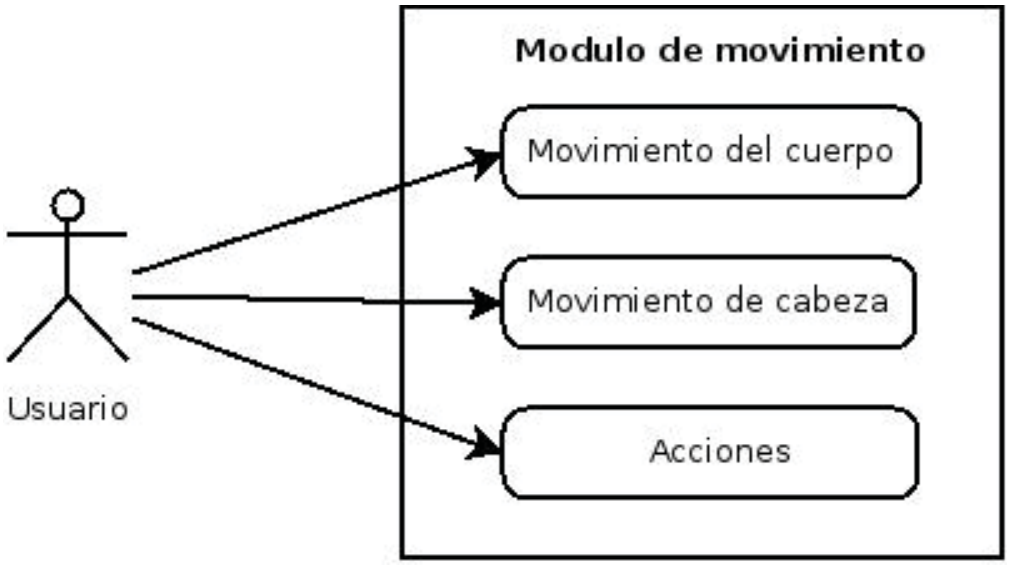

Figura 8: Caso de uso movimiento.

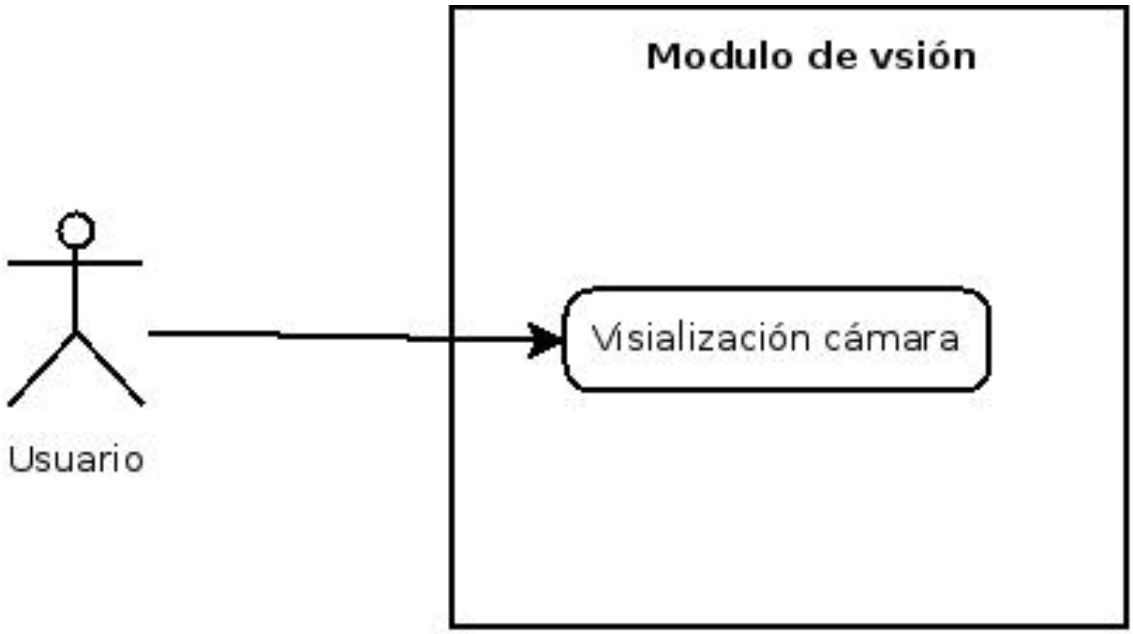

Figura 9: Caso de uso visión.

6.2. Diagrama de componentes

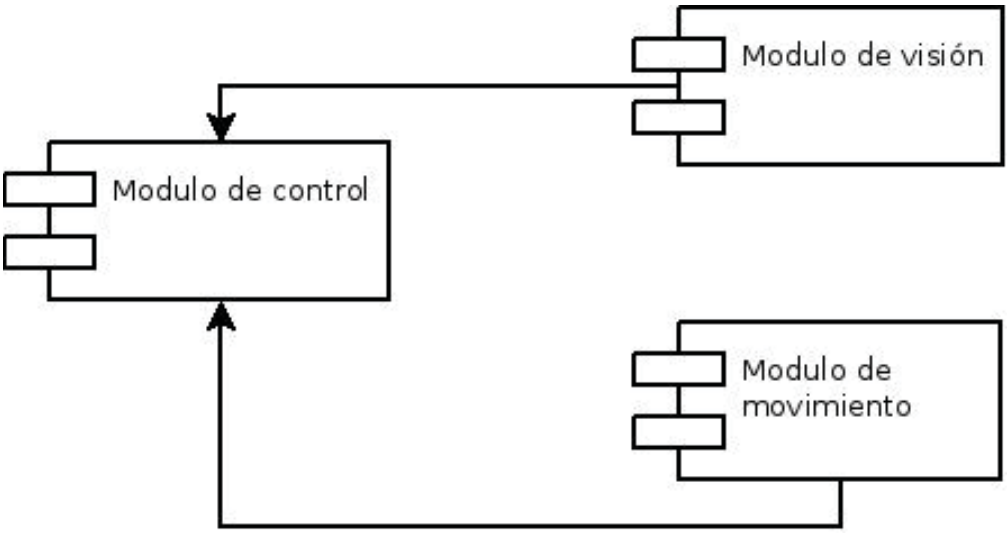

Figura 10: Diagrama de componentes.

6.3. Implementación

La implementación del proyecto completo requirió del lenguaje de programación C++ para el control del robot control a bajo nivel y el lenguaje de programación python para el control a alto nivel del robot a alto nivel por parte del usuario, debido a las diferentes prestaciones que otorga cada lenguaje. Las decisiones tomadas para elegir los lenguajes fueron aprovechando el lenguaje usado por el fabricante del robot para crear su API y la simplicidad de python. Las interfaces que comunican a python con C++ se desarrollaron utilizando el compilador estático Cython.

6.3.1. Modulo de control y Modulo de movimiento

Al implementar el módulo de control se optó por hacer una clase en C++ que permitiera inicializar los controladores linuxCm730 y Cm730, encargados de operar el hardware del robot, de esta manera se puede ordenar a los sensores y actuadores que funcionen, realizando las tareas que se programen.

El siguiente código corresponde a la clase movimiento-Control, que es la responsable de la inicialización y control de los ya mencionados controladores de hardware, es decir controlar al robot a bajo nivel. Esta clase también contiene al módulo de movimiento, siendo sus métodos los

encargados de operar y controlar los movimientos del robot a alto nivel con ayuda de python, el código se muestra en el anexo 10.2.

```
#ifndef CONTROLADORMOVIMIENTO__
#define CONTROLADORMOVIMIENTO__

class controladorMovimiento
{
    bool gestorInicializado;
    bool editorAccionInicializado;
    bool editorAccionEstabaIni;
    bool caminarInicializado;
    bool cabezaInicializada;
    void *linux_cm730;
    void *cm730;
    void *ini;
public:
    controladorMovimiento();
    ~controladorMovimiento();
    bool iniGestorMovimiento();
    void iniEditorAccion();
    void iniCaminar();
    void iniCabeza();
    void ejecutarPagina(int);
    bool ejecutandoAccion();
    void caminar(double, double);
    void caminar(double, double, double);
    void detenerCaminar();
    double getAnguloLimiteSuperiorCabeza();
    double getAnguloLimiteInferiorCabeza();
    double getAnguloLimiteIzquierdoCabeza();
    double getAnguloLimiteDerechoCabeza();
    double getAnguloGiroCabeza();
    double getAnguloInclinacionCabeza();
    void moverCabezaAInicio();
    void moverCabezaPorAngulo(double, double);
    void moverCabezaPorDesplazamiento(double, double);
    void reproducirVoz(char* );
    void cargarActionScript();

protected:
    static void changeCurrentDir();
    static void sighandler(int);
};
#endif
```

Los métodos de la clase movimiento-Control son detallados en el código anexo 10.1.

6.3.2. Modulo de visión

Al implementar el módulo de visión se optó por hacer una clase en C++ que permitiera inicializar los controladores de video, encargados de operar la cámara del robot.

El siguiente código corresponde a la clase controlador-Cámara, que es la responsable de la inicialización y control de los ya mencionados controladores de hardware, es decir controlar al robot a bajo nivel. Esta clase es la encargada de operar y controlar la visualización a alto nivel de las imágenes capturadas por la cámara con ayuda de python, el código se muestra en el anexo 10.4.

```
#ifndef CONTROLADORCAMARA__
#define CONTROLADORCAMARA__

class controladorCamara
{
    void *ini;
    void *streamer;
    void *rgb_output;
    char cwd[1024];
```

```
public:
    controladorCamara();
    ~controladorCamara();
    void streaming();
protected:


};
#endif
```

Los métodos de la clase cámara-Control son detallados en el código anexo 10.3

6.3.3. Makefile, setup.py, www y utilerias

Para poder compilar los códigos desarrollados se hicieron los siguientes archivos:

El archivo Makefile, permite enlazar las bibliotecas necesarias para compilar los códigos en C++.

```
TARGET = controladorMovimiento

INCLUDE_DIRS = -I /darwin/Linux/include -I /darwin/Framework/include

#LDFLAGS= -m32 -L/usr/lib32
CXX = g++
CXXFLAGS +=  -O2 -g -DLINUX -Wall $(INCLUDE_DIRS)
#CXXFLAGS += -O2 -DLINUX -DDEBUG -Wall $(INCLUDE_DIRS)
LFLAGS += -lpthread -ljpeg -lrt

OBJECTS =   main.o controladorMovimiento.o controladorCamara.o StatusCheck.o VisionMode.o


all: $(TARGET)

clean:
  rm -f *.a *.o $(TARGET) core *~ *.so *.lo

darwin.a:
  make -C /darwin/Linux/build

$(TARGET): darwin.a $(OBJECTS)
  $(CXX) $(CFLAGS) $(LFLAGS) $(OBJECTS) /darwin/Linux/lib/darwin.a -o $(TARGET)
  chmod 755 $(TARGET)

# Hace un respaldo "make␣tgz"
tgz: clean
  mkdir -p backups
  tar czvf ./backups/camera_`date +"%Y_%m_%d_%H.%M.%S"`.tgz --exclude backups *
```

El archivo Setup.py, permite compilar con python los códigos en python asociados a los códigos en c++.

```
from distutils.core import setup, Extension
from Cython.Build import cythonize
from Cython.Distutils import build_ext

setup(name='controladorMovimiento',
      version='0.0.1',
      description='DARwIn-OP␣clase␣controladora',
      author='␣',
      author_email='␣',
      url='␣',
      ext_modules=[
            Extension('controladorMovimiento', ['ControladorMovimiento.pyx', 'controladorMovimiento.cpp'],
                language='c++',
                extra_objects=['/darwin/Linux/lib/darwin.a'], # Pasa la biblioteca Robotis darwin.a al linker
                libraries=['jpeg', 'rt'], # Link con las bibliotecas jpeg y rt
```

```
            include_dirs=['/darwin/Framework/include', '/darwin/Linux/include',] # Incluye los archivos para DARwIn-OP de
                 Robotis
            ),  Extension('controladorCamara', ['ControladorCamara.pyx', 'controladorCamara.cpp', ],
            language='c++',
            extra_objects=['/darwin/Linux/lib/darwin.a'], # Pasa la biblioteca
            libraries=['jpeg', 'rt'], # Link con las bibliotecas jpeg y rt
            include_dirs=['/darwin/Framework/include', '/darwin/Linux/include',]
            )

      ],
cmdclass={'build_ext':build_ext}
)
```

6.3.4. Modulo matemático

Este módulo se encarga de realizar cálculos entre puntos en planos y espacio 2D y 3D para determinar distancias y posicionamiento del robot y objetos que lo rodean en su entorno. Se vale del uso de vectores y matrices para lograrlo. Es un módulo auxiliar para los otros tres módulos. Este módulo no fue desarrollado, se optó por hacer uso del módulo matemático proporcionado por el fabricante del robot por su complejidad y para aprovechar el rendimiento que supone al estar hecho en lenguaje C.

7. Resultados

Al finalizar el proyecto se logró obtener una API en python capaz de controlar al robot Darwin-OP. La figura 11 muestra el diagrama de clases con la API en python ya implementada y su relación con el framework proporcionado por el fabricante 5.

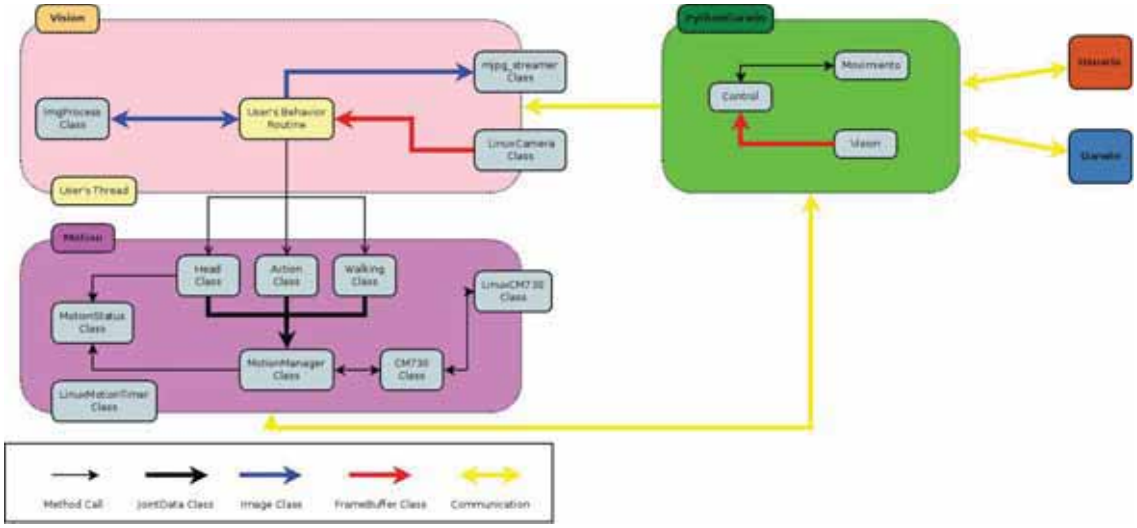

Figura 11: Diagrama de clases de la API en python.

La API es utilizada importando en un código fuente python los módulos controlador-Movimiento y controlador-Cámara.

A continuación se muestra el código de dos pruebas donde se importan los mencionados módulos; en la primera prueba se verificó la movilidad de las extremidades y cabeza del robot, así mismo se le ordena caminar y realizar acciones predefinidas por el fabricante.

```
from __future__ import print_function, division

import time

import controladorMovimiento

controladorDarwin = controladorMovimiento.PycontroladorMovimiento()

inicializado = False

# Intenta inicializar el Motion Manager
try:
    inicializado = controladorDarwin.iniGestorMovimiento()
except:
    print('\nOcurrio una excepción')

if inicializado:
    print("Inicializado")
    controladorDarwin.iniEditorAccion()
    controladorDarwin.ejecutarPagina(15)
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

        # Demostración de caminar
    controladorDarwin.iniCaminar()
```

```
    time.sleep(1)
    controladorDarwin.caminar(5, 0, 0)
    time.sleep(1)
    controladorDarwin.caminar(5, 10)
    time.sleep(1)
    controladorDarwin.caminar(5, -10)
    time.sleep(1)
    controladorDarwin.caminar(-1, 0, 2)

    time.sleep(1)
    controladorDarwin.detenerCaminar()
    time.sleep(1)

    #Demostración de ActionEditor
    time.sleep(0.5)
    controladorDarwin.iniEditorAccion()

    #Espera para terminar antes de salir

    controladorDarwin.ejecutarPagina(16)

    # Espera para terminar antes de salir
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

    controladorDarwin.ejecutarPagina(1)
    while controladorDarwin.ejecutandoAccion():
        time.sleep(10)

    controladorDarwin.ejecutarPagina(15)

    # Espera para terminar antes de salir
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

    # Demostración de la cabeza
    controladorDarwin.iniCabeza()
    controladorDarwin.moverCabezaAInicio()
    time.sleep(1)
    (l, r, u, d) = controladorDarwin.getAnguloLimitesCabeza()
    (p, t) = controladorDarwin.getAngulosGiroInclinacionCabeza()
    print('\nLimits:\n␣l␣=␣%d\n␣r␣=␣%d\n␣u␣=␣%d\n␣d␣=␣%d' % (l, r, u, d))
    print('\nPan␣=␣%d\nTilt␣=␣%d' % (p, t))
    controladorDarwin.moverCabeza(30, 10, modo='directo')
    time.sleep(1)
    (p, t) = controladorDarwin.getAngulosGiroInclinacionCabeza()
    print('\nPan␣=␣%d\nTilt␣=␣%d' % (p, t))

    #Camina y luego mueve la cabeza
    controladorDarwin.iniCaminar()
    time.sleep(1)
    controladorDarwin.moverCabeza(20, -10, modo='directo')
    controladorDarwin.caminar(5, 0, 0)
    time.sleep(5)
    controladorDarwin.detenerCaminar()
    time.sleep(2)
    controladorDarwin.iniEditorAccion()
    time.sleep(2)
    controladorDarwin.ejecutarPagina(15)  # Espera para terminar antes de salir
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)


else:
    print('Inicialización␣Fallo')
```

En la segunda prueba se verifica el video del robot.

```
from __future__ import print_function, division

import time

import controladorMovimiento
import controladorCamara
controladorDarwin = controladorMovimiento.PycontroladorMovimiento()
camaraDarwin = controladorCamara.PycontroladorCamara()

inicializado = False

# Intenta inicializar el Motion Manager
try:
    inicializado = controladorDarwin.iniGestorMovimiento()
except:
    print('\nOcurrio␣una␣excepción')
```

```
if inicializado:
    print("Inicializado")
    controladorDarwin.iniEditorAccion()
    controladorDarwin.ejecutarPagina(15)
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)
    controladorDarwin.ejecutarPagina(16)
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

    controladorDarwin.ejecutarPagina(41)
    controladorDarwin.reproducirVoz("/darwin/Data/mp3/Introduction.mp3")
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

    controladorDarwin.ejecutarPagina(15)
    while controladorDarwin.ejecutandoAccion():
        time.sleep(5)

    while 1:
        camaraDarwin.streaming()
```

8. Análisis y discusión de resultados

Las pruebas realizadas para cada uno de los experimentos fueron hechas con base a las especificaciones requeridas para la aprobación de término de este proyecto. Todas las pruebas fueron realizadas satisfactoriamente.

Algunas cuestiones que deben de ser tomadas en cuenta para futuros proyectos, ya sea de continuación o de nuevas implementaciones, son presentadas a continuación:

- Se puede ampliar el catálogo de acciones predefinidas mediante el uso de la herramienta Action Editor proporcionada por el fabricante.
- Es posible construir paquetes de python que contengan módulos con rutinas del robot que se empleen con regularidad, por ejemplo una rutina que mediante hilos mantengan la cámara transmitiendo de forma continua.
- Por su diseño la API puede ser ampliada mediante código en python o código en C++. La elección del lenguaje dependerá de la finalidad deseada, si se busca rendimiento C++ sería la opción adecuada.
- Las pruebas fueron repetidas al menos 50 veces cada una, para detectar algún fallo en el sistema. Los fallos encontrados fueron corregidos satisfactoriamente.
- Cada prueba tuvo una duración aproximada de 5 minutos, desde el momento en que se iniciaba el sistema hasta que la misma concluía. Esto equivale a un aproximado de 8 horas de pruebas. El resto de las horas calendarizadas para pruebas fueron utilizadas en la corrección de los fallos detectados.

- Dada las condiciones del robot, la mitad de las pruebas se hicieron con Darwin conectado a la corriente de manera directa y la otra mitad se hicieron con las baterías que proporciona el fabricante. Esto demuestra que el sistema es capaz de trabajar tanto con el robot conectado a la corriente como con baterías.
- Todas las pruebas fueron realizadas con una superficie plana y lisa, en el área provista por el asesor para trabajar en este proyecto.
- Algunas de las primeras pruebas resultaban con fallos en la manipulación del robot debido a un problema de calentamiento del robot. Una vez detectado este problema se optó por hacer sesiones de descanso para el robot por cada hora de pruebas. Esto evitó el calentamiento del robot así como fallos en el sistema.

9. Conclusiones

El trabajo de desarrollo e implementación de la API en python ha cumplido con el objetivo principal de este proyecto. Las pruebas realizadas han sido satisfactorias y concuerdan con lo estipulado en la propuesta de este proyecto de integración.

Las pruebas realizadas validaron el uso de la API al controlar el movimiento de cabeza y extremidades del robot. El control de la cámara del robot fue también validado positivamente.

La API permite desarrollar programas para proyectos futuros que hagan uso del robot, es extensible a alto y bajo nivel y facilitará la programación futura del robot. Los programas pueden ser para operar al robot de forma autónoma o de forma guiada, eso responderá a las necesidades del proyecto a desarrollar.

En relación a este proyecto, aún hay bastante trabajo por realizar para futuros proyectos asociados. Se puede ampliar todavía más este proyecto, construyendo con esta API bibliotecas en python para ampliar su gama de funcionalidad a alto nivel o ampliando la misma API a bajo nivel. La robótica es un campo relativamente joven, así que aún hay mucho que hacer e investigar. Es por ello que contar con una herramienta educativa y de investigación como Darwin-Op en nuestra universidad es un gran privilegio.

Referencias

[1] Robotis, [Web en línea]. <>. `support.robotis.com/en/techsupport_eng.htm#product/darwin-op.htm`. [Consulta 10/07/2014]

[2] Cython, [documentación en línea]. <>. `docs.cython.org`. [Consulta 10/07/2014]

[3] Python, [documentación en línea]. <>. `docs.python.org`. [Consulta 10/07/2014]

[4] A. Ollero Baturone, *Robótica: manipulación y robots móviles*, Marcombo, España, 2001.

[5] John. J. Craig, *Robótica*, Pearson Educación, 3a Edición, 2006.

[6] M. Lutz, *Python programming*,O'Reilly Media, Inc., 2006.

[7] L. Van Lancker, *HTML5 y CSS3: Domine los estándares de las aplicaciones Web*, Ediciones ENI, 2a. Edición, 2013.

10. Código fuente

A continuación se muestra el código fuente de las partes que componen la API en python que permite el control del robot humanoide Darwin-OP.

10.1. Clase encargada del módulo de control y módulo de movimiento controladorMovimiento.cpp

```
/*
 * Esta clase permite acceso a alto nivel y control de los movimientos de Robotis Darwin-OP
 */


#include <unistd.h>
#include <string.h>
#include <string>
#include <libgen.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <pthread.h>

#include "Point.h"
#include "mjpg_streamer.h"

#include "minIni.h"
#include "Action.h"
#include "Head.h"
#include "Walking.h"
#include "MX28.h"
#include "MotionManager.h"
#include "LinuxMotionTimer.h"
#include "LinuxCM730.h"
#include "LinuxActionScript.h"

#include "controladorMovimiento.h"

#ifdef MX28_1024
#define MOTION_FILE_PATH    "/darwin/Data/motion_1024.bin"
#else
#define MOTION_FILE_PATH    "/darwin/Data/motion_4096.bin"
#endif

#define INI_FILE_PATH       "/darwin/Data/config.ini"
#define SCRIPT_FILE_PATH    "script.asc"

#define PORT 9930
#define BUFLEN 70
#define SAMPLE_RATE (44100)

controladorMovimiento::controladorMovimiento()
/* Constructor */
{
    gestorInicializado = false;         // Gestor Movimiento no esta inicializado
    editorAccionInicializado = false;    // Editor Acción no esta inicializado
    editorAccionEstabaIni = false;
    caminarInicializado = false;         // Caminar no esta inicializado
    cabezaInicializada = false;       // Cabeza no esta inicializado
    linux_cm730 = new LinuxCM730("/dev/ttyUSB0"); // Crea objetos para el controlador linux CM730
    cm730 = new CM730(static_cast<LinuxCM730*> (linux_cm730));     // Crea un nuevo objeto para el controlador CM730
    ini = new minIni(INI_FILE_PATH);

    // Signal handlers
    signal(SIGABRT, &controladorMovimiento::sighandler);
    signal(SIGTERM, &controladorMovimiento::sighandler);
    signal(SIGQUIT, &controladorMovimiento::sighandler);
    signal(SIGINT, &controladorMovimiento::sighandler);
}
```

```
controladorMovimiento::~controladorMovimiento()
/* Destructor */
{
}

bool controladorMovimiento::iniGestorMovimiento()
/*
 * Inicializa el motion manager para controlar a DARwIn-OP.
 */
{
    int MAX_INTENTOS = 5; // Número de intentos para inicializar
    int numIntento = 0;

    controladorMovimiento::changeCurrentDir();

    if(Action::GetInstance()->LoadFile(MOTION_FILE_PATH) == false)
    {
        printf("Fallo la carga del archivo de movimiento %s", MOTION_FILE_PATH);
        return false;
    }

    // Intenta inicializar varias veces
    while((numIntento < MAX_INTENTOS) && (gestorInicializado == false))
    {
        if(MotionManager::GetInstance()->Initialize(static_cast<CM730*> (cm730)) == true)
        {
            gestorInicializado = true; // Indica que manager se ha inicializado
        }
        else
        {
            numIntento++;
            printf("Fallo inicialización, intento %i", numIntento);
            usleep(1000000); // Tiempo de espera antes de intentar inicializar otra vez
        }

    }

    // Si se inicializo empieza el timer
    if (gestorInicializado)
    {
        LinuxMotionTimer *motion_timer = new LinuxMotionTimer(MotionManager::GetInstance());
        motion_timer->Start();
    }

    // Inicializa  a la posicion de inicio
    MotionManager::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));
    Action::GetInstance()->m_Joint.SetEnableBody(true, true);
    MotionManager::GetInstance()->SetEnable(true);


    return gestorInicializado;
}

void controladorMovimiento::iniEditorAccion()
{
    /*
     * Hace uso del editor de acción para controlar a DARwIn-OP
     */
    if(gestorInicializado)
    {
        if(editorAccionInicializado == false)
        {
            MotionManager::GetInstance()->AddModule((MotionModule*)Action::GetInstance());
            MotionManager::GetInstance()->SetEnable(true);
            //Indica que editor de acción se ha inicilizado
            editorAccionInicializado = true;
        }

    }
    else
    {
        printf("Gestor de Movimiento no inicializado. Ejecutar iniGestorMovimiento() primero");
    }
}

void controladorMovimiento::iniCaminar()
{
    /*
     * Prepara a Darwin-OP para caminar
     */


    if(gestorInicializado)
    {
        if(editorAccionInicializado)
        {
```

```
            editorAccionEstabaIni=editorAccionInicializado;
            editorAccionInicializado=false;

        }
        Walking::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));

        MotionManager::GetInstance()->AddModule((MotionModule*)Walking::GetInstance());

        int n = 0;
        int param[JointData::NUMBER_OF_JOINTS * 5];
        int cMetaPosicion, cPosicionInicio, cDistancia;

        for(int id=JointData::ID_R_SHOULDER_PITCH; id<JointData::NUMBER_OF_JOINTS; id++)
        {
            cPosicionInicio = MotionStatus::m_CurrentJoints.GetValue(id);
            cMetaPosicion = Walking::GetInstance()->m_Joint.GetValue(id);
            if( cPosicionInicio > cMetaPosicion )
                cDistancia = cPosicionInicio - cMetaPosicion;
            else
                cDistancia = cMetaPosicion - cPosicionInicio;

            cDistancia >>= 2;
            if( cDistancia < 8 )
                cDistancia = 8;

            param[n++] = id;
            param[n++] = CM730::GetLowByte(cMetaPosicion);
            param[n++] = CM730::GetHighByte(cMetaPosicion);
            param[n++] = CM730::GetLowByte(cDistancia);
            param[n++] = CM730::GetHighByte(cDistancia);
        }
        (static_cast<CM730*>(cm730))->SyncWrite(MX28::P_GOAL_POSITION_L, 5, JointData::NUMBER_OF_JOINTS - 1, param);

        usleep(1000000); // Espera un tiempo para que Darwin-OP se coloque en la posición

        // Habilita caminar y el gestor de movimiento
        Walking::GetInstance()->m_Joint.SetEnableBody(true);
        MotionManager::GetInstance()->SetEnable(true);


        Walking::GetInstance()->Initialize();

        // Indica que walking se ha inicializado
        caminarInicializado = true;
    }

    else
    {
        printf("Gestor␣de␣Movimiento␣no␣inicializado.␣Ejecutar␣iniGestorMovimiento()␣primero");
    }
}

void controladorMovimiento::iniCabeza()
{
    /*
     * Movimiento de cabeza Darwin-OP
     */

    if(gestorInicializado)
    {
        if(editorAccionInicializado)
        {
            editorAccionInicializado=false;

        }
        MotionManager::GetInstance()->AddModule((MotionModule*)Head::GetInstance());
        Head::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));
        Head::GetInstance()->Initialize();

        MotionManager::GetInstance()->SetEnable(true);
        cabezaInicializada = true;
    }

    else
    {
        printf("Gestor␣de␣Movimiento␣no␣inicializado.␣Ejecutar␣iniGestorMovimiento()␣primero");
    }
}

void controladorMovimiento::moverCabezaAInicio()
/* Mueve la cabeza a la posicion inicial*/
{
    if(cabezaInicializada)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveToHome();
```

```
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
    }
}

void controladorMovimiento::moverCabezaPorAngulo(double giro, double inclinacion)
/*
 * Mueve la cabeza a angulos de giro e inclinación especificos
 *    giro  - angulo de giro
 *    inclinacion - angulo de inclinación
 */
{
    if(cabezaInicializada)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveByAngle(giro, inclinacion);
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
    }
}

void controladorMovimiento::moverCabezaPorDesplazamiento(double giro, double inclinacion)
/*
 * Desplaza  la cabeza desde la posición actual el grado de angulos especificados
 *
 */
{
    if(cabezaInicializada)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveByAngleOffset(giro, inclinacion);
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
    }
}

double controladorMovimiento::getAnguloLimiteSuperiorCabeza()
/* Ángulo límite superior de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetTopLimitAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteInferiorCabeza()
/* Ángulo límite inferior de la cabeza*/
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetBottomLimitAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteIzquierdoCabeza()
/* Ángulo límite izquierdo de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetLeftLimitAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteDerechoCabeza()
```

```
/* Ángulo límite derecho de la cabeza*/
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetRightLimitAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloGiroCabeza()
/* Ángulo de giro actual de la cabeza*/
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetPanAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloInclinacionCabeza()
/* Ángulo de inclinación actual de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetTiltAngle();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
        return 0.0;
    }
}

void controladorMovimiento::ejecutarPagina(int numPagina)
/* Ejecuta el número de página especificado en el editor de acción */
{

    if(editorAccionInicializado)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        Action::GetInstance()->Start(numPagina);
    }
    else
    {
        printf("Editor de accion no inicializado, ejecutar iniEditorAccion()");
    }
}

bool controladorMovimiento::ejecutandoAccion()
/*
 * Verifica si la acción aun se esta ejecutando
 */
{
    return Action::GetInstance()->IsRunning();
}

void controladorMovimiento::caminar(double duracion, double direccion)
/*
 * Camina la duración especificada en la direccion dada
 *
 */
{
    if(caminarInicializado)
    {
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->A_MOVE_AMPLITUDE = direccion;
        Walking::GetInstance()->Start();

        if(duracion >= 0.0)
        {
            usleep(duracion*1000000);
            Walking::GetInstance()->Stop();
        }
    }
```

```
    else
    {
        printf("Caminar no inicializado, ejecutar iniCaminar()");
    }

}

void controladorMovimiento::caminar(double duracion, double direccion, double tamPaso)
/*
 * Camina en la dirección especificada la cantidad de tiempo indicada con cierto tamaño de paso
 *
 */
{
    if(caminarInicializado)
    {
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->A_MOVE_AMPLITUDE = direccion;
        Walking::GetInstance()->X_MOVE_AMPLITUDE = tamPaso;
        Walking::GetInstance()->Start();

        if(duracion >= 0.0)
        {
            usleep(duracion*1000000);
            Walking::GetInstance()->Stop();
        }
    }
    else
    {
        printf("Caminar no inicializado, ejecutar iniCaminar()");
    }

}

void controladorMovimiento::detenerCaminar()
/*
 * Deja de caminar
 */
{
    if(caminarInicializado)
    {
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->Stop();
        if(editorAccionEstabaIni)
        {
            editorAccionEstabaIni = false;
            iniEditorAccion();
        }
    }
    else
    {
        printf("Caminar no inicializado, ejecutar iniCaminar()");
    }
}


void controladorMovimiento::changeCurrentDir()
{

    char exepath[1024] = {0};
    if(readlink("/proc/self/exe", exepath, sizeof(exepath)) != -1)
        chdir(dirname(exepath));
}

void controladorMovimiento::sighandler(int sig)
{
    /*
    * Signal handling
    */
    struct termios term;
    tcgetattr( STDIN_FILENO, &term );
    term.c_lflag |= ICANON | ECHO;
    tcsetattr( STDIN_FILENO, TCSANOW, &term );

    exit(0);
}

void controladorMovimiento::reproducirVoz(char* archivomp3)
```

```
{

    LinuxActionScript::PlayMP3Wait(archivomp3);

}

void controladorMovimiento::cargarActionScript()
{
    if(LinuxActionScript::m_is_running == 0)
    {
        LinuxActionScript::ScriptStart(SCRIPT_FILE_PATH);
    }
}
```

10.2. Clase wrapper en python para el módulo de control y módulo de movimiento ControladorMovimiento.pyx

```
/*
 * Esta clase permite acceso a alto nivel y control de los movimientos de Robotis Darwin-OP
 */


#include <unistd.h>
#include <string.h>
#include <string>
#include <libgen.h>
#include <stdlib.h>
#include <stdio.h>
#include <time.h>
#include <signal.h>
#include <termios.h>
#include <sys/types.h>
#include <sys/socket.h>
#include <arpa/inet.h>
#include <netinet/in.h>
#include <pthread.h>

#include "Point.h"
#include "mjpg_streamer.h"

#include "minIni.h"
#include "Action.h"
#include "Head.h"
#include "Walking.h"
#include "MX28.h"
#include "MotionManager.h"
#include "LinuxMotionTimer.h"
#include "LinuxCM730.h"
#include "LinuxActionScript.h"

#include "controladorMovimiento.h"

#ifdef MX28_1024
#define MOTION_FILE_PATH    "/darwin/Data/motion_1024.bin"
#else
#define MOTION_FILE_PATH    "/darwin/Data/motion_4096.bin"
#endif

#define INI_FILE_PATH       "/darwin/Data/config.ini"
#define SCRIPT_FILE_PATH    "script.asc"

#define PORT 9930
#define BUFLEN 70
#define SAMPLE_RATE (44100)

controladorMovimiento::controladorMovimiento()
/* Constructor */
{
    gestorInicializado = false;         // Gestor Movimiento no esta inicializado
    editorAccionInicializado = false;    // Editor Acción no esta inicializado
    editorAccionEstabaIni = false;
    caminarInicializado = false;         // Caminar no esta inicializado
    cabezaInicializada = false;       // Cabeza no esta inicializado
    linux_cm730 = new LinuxCM730("/dev/ttyUSB0"); // Crea objetos para el controlador linux CM730
    cm730 = new CM730(static_cast<LinuxCM730*> (linux_cm730));     // Crea un nuevo objeto para el controlador CM730
    ini = new minIni(INI_FILE_PATH);

    // Signal handlers
    signal(SIGABRT, &controladorMovimiento::sighandler);
    signal(SIGTERM, &controladorMovimiento::sighandler);
    signal(SIGQUIT, &controladorMovimiento::sighandler);
    signal(SIGINT, &controladorMovimiento::sighandler);
}

controladorMovimiento::~controladorMovimiento()
/* Destructor */
{
}

bool controladorMovimiento::iniGestorMovimiento()
/*
 * Inicializa el motion manager para controlar a DARwIn-OP.
 */
{
    int MAX_INTENTOS = 5; // Número de intentos para inicializar
    int numIntento = 0;

    controladorMovimiento::changeCurrentDir();
```

```
    if(Action::GetInstance()->LoadFile(MOTION_FILE_PATH) == false)
    {
        printf("Fallo␣la␣carga␣del␣archivo␣de␣movimiento␣%s", MOTION_FILE_PATH);
        return false;
    }

    // Intenta inicializar varias veces
    while((numIntento < MAX_INTENTOS) && (gestorInicializado == false))
    {
        if(MotionManager::GetInstance()->Initialize(static_cast<CM730*> (cm730)) == true)
        {
            gestorInicializado = true; // Indica que manager se ha inicializado
        }
        else
        {
            numIntento++;
            printf("Fallo␣inicialización,␣intento␣%i", numIntento);
            usleep(1000000); // Tiempo de espera antes de intentar inicializar otra vez
        }

    }

    // Si se inicializo empieza el timer
    if (gestorInicializado)
    {
        LinuxMotionTimer *motion_timer = new LinuxMotionTimer(MotionManager::GetInstance());
        motion_timer->Start();
    }

    // Inicializa  a la posicion de inicio
    MotionManager::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));
    Action::GetInstance()->m_Joint.SetEnableBody(true, true);
    MotionManager::GetInstance()->SetEnable(true);


    return gestorInicializado;
}

void controladorMovimiento::iniEditorAccion()
{
    /*
     * Hace uso del editor de acción para controlar a DARwIn-OP
     */
    if(gestorInicializado)
    {
        if(editorAccionInicializado == false)
        {
            MotionManager::GetInstance()->AddModule((MotionModule*)Action::GetInstance());
            MotionManager::GetInstance()->SetEnable(true);
            //Indica que editor de acción se ha inicilizado
            editorAccionInicializado = true;
        }

    }
    else
    {
        printf("Gestor␣de␣Movimiento␣no␣inicializado.␣Ejecutar␣iniGestorMovimiento()␣primero");
    }
}

void controladorMovimiento::iniCaminar()
{
    /*
     * Prepara a Darwin-OP para caminar
     */


    if(gestorInicializado)
    {
        if(editorAccionInicializado)
        {
            editorAccionEstabaIni=editorAccionInicializado;
            editorAccionInicializado=false;

        }
        Walking::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));

        MotionManager::GetInstance()->AddModule((MotionModule*)Walking::GetInstance());

        int n = 0;
        int param[JointData::NUMBER_OF_JOINTS * 5];
        int cMetaPosicion, cPosicionInicio, cDistancia;

        for(int id=JointData::ID_R_SHOULDER_PITCH; id<JointData::NUMBER_OF_JOINTS; id++)
        {
            cPosicionInicio = MotionStatus::m_CurrentJoints.GetValue(id);
```

```
            cMetaPosicion = Walking::GetInstance()->m_Joint.GetValue(id);
            if( cPosicionInicio > cMetaPosicion )
                cDistancia = cPosicionInicio - cMetaPosicion;
            else
                cDistancia = cMetaPosicion - cPosicionInicio;

            cDistancia >>= 2;
            if( cDistancia < 8 )
                cDistancia = 8;

            param[n++] = id;
            param[n++] = CM730::GetLowByte(cMetaPosicion);
            param[n++] = CM730::GetHighByte(cMetaPosicion);
            param[n++] = CM730::GetLowByte(cDistancia);
            param[n++] = CM730::GetHighByte(cDistancia);
        }
        (static_cast<CM730*>(cm730))->SyncWrite(MX28::P_GOAL_POSITION_L, 5, JointData::NUMBER_OF_JOINTS - 1, param);

        usleep(1000000); // Espera un tiempo para que Darwin-OP se coloque en la posición

        // Habilita caminar y el gestor de movimiento
        Walking::GetInstance()->m_Joint.SetEnableBody(true);
        MotionManager::GetInstance()->SetEnable(true);


        Walking::GetInstance()->Initialize();

        // Indica que walking se ha inicializado
        caminarInicializado = true;
    }

    else
    {
        printf("Gestor de Movimiento no inicializado. Ejecutar iniGestorMovimiento() primero");
    }
}

void controladorMovimiento::iniCabeza()
{
    /*
     * Movimiento de cabeza Darwin-OP
     */

    if(gestorInicializado)
    {
        if(editorAccionInicializado)
        {
            editorAccionInicializado=false;

        }
        MotionManager::GetInstance()->AddModule((MotionModule*)Head::GetInstance());
        Head::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));
        Head::GetInstance()->Initialize();

        MotionManager::GetInstance()->SetEnable(true);
        cabezaInicializada = true;
    }

    else
    {
        printf("Gestor de Movimiento no inicializado. Ejecutar iniGestorMovimiento() primero");
    }
}

void controladorMovimiento::moverCabezaAInicio()
/* Mueve la cabeza a la posicion inicial*/
{
    if(cabezaInicializada)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveToHome();
    }
    else
    {
        printf("Cabeza no inicializada, ejecutar iniCabeza()");
    }
}

void controladorMovimiento::moverCabezaPorAngulo(double giro, double inclinacion)
/*
 * Mueve la cabeza a angulos de giro e inclinación especificos
 *    giro  - angulo de giro
 *    inclinacion - angulo de inclinación
 */
{
    if(cabezaInicializada)
```

```
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveByAngle(giro, inclinacion);
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
    }
}

void controladorMovimiento::moverCabezaPorDesplazamiento(double giro, double inclinacion)
/*
 * Desplaza  la cabeza desde la posición actual el grado de angulos especificados
 *
 */
{
    if(cabezaInicializada)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(false);
        MotionStatus::m_CurrentJoints.SetEnableHeadOnly(true);
        Head::GetInstance()->MoveByAngleOffset(giro, inclinacion);
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
    }
}

double controladorMovimiento::getAnguloLimiteSuperiorCabeza()
/* Ángulo límite superior de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetTopLimitAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteInferiorCabeza()
/* Ángulo límite inferior de la cabeza*/
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetBottomLimitAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteIzquierdoCabeza()
/* Ángulo límite izquierdo de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetLeftLimitAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloLimiteDerechoCabeza()
/* Ángulo límite derecho de la cabeza*/
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetRightLimitAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloGiroCabeza()
/* Ángulo de giro actual de la cabeza*/
```

```
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetPanAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

double controladorMovimiento::getAnguloInclinacionCabeza()
/* Ángulo de inclinación actual de la cabeza */
{
    if(cabezaInicializada)
    {
        return Head::GetInstance()->GetTiltAngle();
    }
    else
    {
        printf("Cabeza␣no␣inicializada,␣ejecutar␣iniCabeza()");
        return 0.0;
    }
}

void controladorMovimiento::ejecutarPagina(int numPagina)
/* Ejecuta el número de página especificado en el editor de acción */
{

    if(editorAccionInicializado)
    {
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        Action::GetInstance()->Start(numPagina);
    }
    else
    {
        printf("Editor␣de␣accion␣no␣inicializado,␣ejecutar␣iniEditorAccion()");
    }
}

bool controladorMovimiento::ejecutandoAccion()
/*
 * Verifica si la acción aun se esta ejecutando
 */
{
    return Action::GetInstance()->IsRunning();
}

void controladorMovimiento::caminar(double duracion, double direccion)
/*
 * Camina la duración especificada en la direccion dada
 *
 */
{
    if(caminarInicializado)
    {
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->A_MOVE_AMPLITUDE = direccion;
        Walking::GetInstance()->Start();

        if(duracion >= 0.0)
        {
            usleep(duracion*1000000);
            Walking::GetInstance()->Stop();
        }
    }
    else
    {
        printf("Caminar␣no␣inicializado,␣ejecutar␣iniCaminar()");
    }

}

void controladorMovimiento::caminar(double duracion, double direccion, double tamPaso)
/*
 * Camina en la dirección especificada la cantidad de tiempo indicada con cierto tamaño de paso
 *
 */
{
    if(caminarInicializado)
    {
```

```
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->A_MOVE_AMPLITUDE = direccion;
        Walking::GetInstance()->X_MOVE_AMPLITUDE = tamPaso;
        Walking::GetInstance()->Start();

        if(duracion >= 0.0)
        {
            usleep(duracion*1000000);
            Walking::GetInstance()->Stop();
        }
    }
    else
    {
        printf("Caminar␣no␣inicializado,␣ejecutar␣iniCaminar()");
    }

}

void controladorMovimiento::detenerCaminar()
/*
 * Deja de caminar
 */
{
    if(caminarInicializado)
    {
        // Habilita el cuerpo del robot pero si la cabeza esta inicializada no la mueve
        MotionStatus::m_CurrentJoints.SetEnableBody(true);
        if(cabezaInicializada)
        {
            Walking::GetInstance()->m_Joint.SetEnableHeadOnly(false);
        }
        Walking::GetInstance()->Stop();
        if(editorAccionEstabaIni)
        {
            editorAccionEstabaIni = false;
            iniEditorAccion();
        }
    }
    else
    {
        printf("Caminar␣no␣inicializado,␣ejecutar␣iniCaminar()");
    }
}


void controladorMovimiento::changeCurrentDir()
{

    char exepath[1024] = {0};
    if(readlink("/proc/self/exe", exepath, sizeof(exepath)) != -1)
        chdir(dirname(exepath));
}

void controladorMovimiento::sighandler(int sig)
{
    /*
    * Signal handling
    */
    struct termios term;
    tcgetattr( STDIN_FILENO, &term );
    term.c_lflag |= ICANON | ECHO;
    tcsetattr( STDIN_FILENO, TCSANOW, &term );

    exit(0);
}

void controladorMovimiento::reproducirVoz(char* archivomp3)
{

    LinuxActionScript::PlayMP3Wait(archivomp3);

}

void controladorMovimiento::cargarActionScript()
{
    if(LinuxActionScript::m_is_running == 0)
    {
        LinuxActionScript::ScriptStart(SCRIPT_FILE_PATH);
    }
}
```

10.3. Clase encargada del módulo de visión controladorCamara.cpp

```
#include <stdio.h>

#include <unistd.h>

#include <limits.h>

#include <string.h>

#include <libgen.h>

#include <signal.h>


#include "mjpg_streamer.h"

#include "LinuxDARwIn.h"
#include  "controladorCamara.h"


#include "StatusCheck.h"

#include "VisionMode.h"


#ifdef MX28_1024
#define MOTION_FILE_PATH    "/darwin/Data/motion_1024.bin"
#else
#define MOTION_FILE_PATH    "/darwin/Data/motion_4096.bin"
#endif

#define INI_FILE_PATH       "/darwin/Data/config.ini"

#define SCRIPT_FILE_PATH    "script.asc"


#define U2D_DEV_NAME0       "/dev/ttyUSB0"

#define U2D_DEV_NAME1       "/dev/ttyUSB1"



void change_current_dir()

{

    char exepath[1024] = {0};

    if(readlink("/proc/self/exe", exepath, sizeof(exepath)) != -1)

    {

        if(chdir(dirname(exepath)))

            fprintf(stderr, "chdir␣error!!␣\n");

    }

}


void sighandler(int sig)

{

    exit(0);

}


controladorCamara::controladorCamara()

{

    signal(SIGABRT, &sighandler);

    signal(SIGTERM, &sighandler);

    signal(SIGQUIT, &sighandler);
```

```
    signal(SIGINT, &sighandler);


    ini = new minIni(INI_FILE_PATH);
    rgb_output = new Image(Camera::WIDTH, Camera::HEIGHT, Image::RGB_PIXEL_SIZE);


    LinuxCamera::GetInstance()->Initialize(0);
    LinuxCamera::GetInstance()->SetCameraSettings(CameraSettings());    // set default
    LinuxCamera::GetInstance()->LoadINISettings(static_cast<minIni*> (ini));                   // load from ini
    streamer = new mjpg_streamer(Camera::WIDTH, Camera::HEIGHT);
    httpd::ini = static_cast<minIni*> (ini);

}



controladorCamara::~controladorCamara()
{

}


void controladorCamara::streaming()
{

    LinuxCamera::GetInstance()->CaptureFrame();

    memcpy(static_cast<Image*> (rgb_output)->m_ImageData, LinuxCamera::GetInstance()->fbuffer->m_RGBFrame->m_ImageData, LinuxCamera
          ::GetInstance()->fbuffer->m_RGBFrame->m_ImageSize);



    static_cast<mjpg_streamer*> (streamer)->send_image(static_cast<Image*> (rgb_output));

}
```

10.4. Clase wrapper en python para el módulo de vision ControladorCamara.pyx

```
# distutils: language = c++
# distutils: sources = controladorCamara.cpp

from libcpp cimport bool

cdef extern from "controladorCamara.h":  # Hace disponible la declaracion de la clase

    cdef cppclass controladorCamara:    # Aniade la clase C++
        controladorCamara() except +    # Permite a Cython producir un error de Python si uno ocurre
        void streaming()

# Crea la clase wrapper Python
cdef class PycontroladorCamara:
    cdef controladorCamara *thisptr # Crea un puntero C++ al objeto controladorMovimiento
    def __cinit__(self):
        self.thisptr = new controladorCamara()
    def __dealloc__(self):
        del self.thisptr
    def streaming(self):
        self.thisptr.streaming()
```