FACULTAD DE INGENIERÍA

UNIVERSIDAD DE LA REPÚBLICA URUGUAY

CARA ROBOTICA

Realizado por Joel Cabrera (joel.cabrera@fing.edu.uy)

Introducción

El proyecto comienza siendo un trabajo para la UC 'Taller de Introducción a la Computación' (2021) de la Facultad de Ingeniería. El objetivo fue programar una cara robótica la cual sea capaz de imitar emociones y seguir al humano con los ojos. Una primera versión del proyecto puede encontrarse aquí.

Debido a que se trataba de una unidad curricular de computación, el interés estaba en la programación y no en el hardware, por lo que se buscó un modelo de una cara animatrónica para programarla. En particular, el modelo es el que aparece en el vídeo de James Bruton 'How to Make Robots Move Smoothly'.

A julio de 2024, el robot es capaz de seguir con la mirada a un usuario, ya sea moviendo los iris, girando el cuello y levantando la mirada.

Hardware

Componentes

Para el desarrollo se utilizaron los siguientes componentes:

- Arduino MEGA (1)
- Servo Mg996r (3)
- Micro servo SG90 (6)
- Micro servo MG90S (2)
- Protoshield

Modelo 3D

El modelo 3D de la cara fue creado por James Bruton (a excepción del soporte de las cejas). En la carpeta *stl* pueden encontrarse cada uno de los componentes.

Pines

Motor	**Pin**
Párpado derecho	3
Párpado izquierdo	4

Iris derecho	5
Iris izquierdo	6
Cuello derecho	7
Cuello izquierdo	8
Ceja derecha (1 - más abajo)	9
Ceja derecha (2 - central)	10
Ceja izquierda (1 - más abajo)	11
Ceja izquierda (2 - central)	12
Cuello abajo	13

Software

Se utilizó ROS Noetic para la organización de los componentes del software. Los programas desarrollados, junto con las distintas clases, fueron creados y organizados en archivos .h y .cpp para el código de Arduino, los cuales se encargan del control de los motores. La comunicación entre Arduino y ROS se realizó mediante rosserial_arduino.

En *cara_workspace* puede encontrarse la implementación de los nodos y los mensajes creados. Mientras que las clases creadas para controlar los motores y el programa de Arduino se encuentran en la carpeta *programa_ros_2024*.

Clases

Se presentan las clases creadas y sus atributos.

Motor

La clase *Motor* se utiliza para controlar los motores de un robot. Define varios atributos y métodos que permiten gestionar y controlar el estado y movimiento de los motores.

Atributos

- **int Pin**: pin del motor
- **Servo* motor**: puntero al servo (se necesita la librería *Servo.h*)
- **int neutralPosition**: la posición neutra del motor (robot mirando al frente)

- **int (min|max)Position**: posición (mínima|maxima) que puede alcanzar el robot para que no se tranque ni rompa.
- **float actualPosition**: posición actual del motor.
- **float lastPosition**: posición anterior del motor.
- **float desiredPosition**: posición deseada del motor. Tanto esta como las últimas tres, se utilizan para el control suave y PID.

Ojos

La clase *Ojos* representa el control de los motores asociados con los ojos del robot, incluyendo tanto los motores para los párpados y el iris de cada ojo.

Atributos

- **Motor iris_izq**: Motor que controla el iris izquierdo.
- **Motor iris_der**: Motor que controla el iris derecho.
- **Motor parpado_izq**: Motor que controla el párpado izquierdo.
- **Motor parpado_der**: Motor que controla el párpado derecho.
- **unsigned int cant_motores**: Cantidad de motores controlados por la clase Ojos.

Cuello

La clase *Cuello* representa el control de los motores asociados con el cuello del robot.

Atributos

- **Motor cuello_izq**: Motor que controla el lado izquierdo del cuello.
- **Motor cuello_der**: Motor que controla el lado derecho del cuello.
- **unsigned int cant_motores**: Cantidad de motores controlados por la clase Cuello

Cejas

La clase *Cejas* representa el control de los motores asociados con las cejas del robot. Permite manipular y obtener información sobre los motores que controlan el movimiento de las cejas del robot.

Atributos

- **Motor ceja_izq1**: Motor que controla la parte inferior de la ceja izquierda.
- **Motor ceja_izq2**: Motor que controla la parte superior de la ceja izquierda.
- **Motor ceja_der1**: Motor que controla la parte inferior de la ceja derecha.

- **Motor ceja_der2**: Motor que controla la parte superior de la ceja derecha.
- **unsigned int cant_motores**: Cantidad de motores controlados por la clase Cejas.

Cara

La clase *Cara* representa el conjunto de motores y mecanismos que controlan las expresiones faciales del robot, incluyendo los ojos, el cuello y las cejas. Permite manipular y obtener información sobre estos componentes para realizar movimientos coordinados.

Atributos

- **unsigned int cant_motores**: Cantidad total de motores controlados por la clase Cara.
- ***Ojos* ojos***: Puntero a un objeto Ojos que controla los movimientos de los ojos.
- ***Cuello* cuello*:** Puntero a un objeto Cuello que controla los movimientos del cuello.
- ***Cejas* cejas***: Puntero a un objeto Cejas que controla los movimientos de las cejas.
- ***Motor** **motores[10]**: Arreglo que almacena punteros a los motores controlados por la clase Cara.

Métodos

Por simplicidad, solamente se nombraran algunos de los métodos que podrían llegar a ser confusos.

- **void raise_neck(int degree)**: Levanta el cuello **una cantidad** de grados específicos.
- **void look_to(int degree)**: Mueve los ojos **a un** ángulo específico.
- **void reset_motors()**: Restablece los motores a sus posiciones neutrales.
- **void moveUntilEnd(float alpha)**: Mueve todos los motores de forma suavizada desde su posición actual hasta la posición deseada. El alpha debe ser un real de 0 a 1 y representa la fracción del movimiento que debe realizarse hacia la posición deseada en cada iteración.
- **void move_motor(int pin, int degree)**: Mueve un motor específico a un ángulo determinado basado en el pin.
- **float* getPosIrisActuales()**: Devuelve un puntero a un arreglo con las posiciones actuales de los iris.
- **float* getLimIris()**: Devuelve un puntero a un arreglo con los límites de movimiento de los iris.
- **float getLimMinIris()**: Devuelve el límite mínimo del movimiento del iris.
- **float* getPosCuelloActuales()**: Devuelve un puntero a un arreglo con las posiciones actuales del cuello.
- **float* getLimCuello()**: Devuelve un puntero a un arreglo con los límites de movimiento del cuello, es decir, su posición mínima y máxima.

Detector de caras

Se utilizó la librería face_recognition, la cual detecta la ubicación de caras en una imagen, así como las ubicaciones de cada feature de la cara.

El nodo face_detector_node lee del tópico *camera/rgb/image_raw* la imagen de la cámara y publica en el tópico *face_detector/rectangle* el punto medio de la cara. Algunos comentarios:

- Publica distintos mensajes de todas las caras que detecta. No se probó con más de una cara pero es probable que intente seguir a ambas.
- El punto medio lo representa como un par de números del 1 al 100.
- Se tuvo que realizar un resize de la imagen para que no demore tanto.

```
face_locations = face_recognition.face_locations(frame)
for face_location in face_locations:
    top, right, bottom, left = face_location
    center_of_face = (int((right+left)/2),int((top+bottom)/2))
    x,y = center_of_face
    x = int(round(x/small_frame.shape[1],2)*100)
    y = int(round(y/small_frame.shape[0],2)*100)
```

Seguidor de caras

programa_ros_2024.ino

El seguidor de caras en este sistema está diseñado para ajustar la orientación de los componentes faciales de un robot en respuesta a la posición de una cara detectada en una imagen. Este proceso involucra varios pasos clave:

1) **Recepción de Datos de la Cara**: El sistema recibe datos sobre la posición de la cara a través de un tópico ROS. Estos datos se publican en el tópico */face_detector/rectangle*, que contiene la posición horizontal (x) y vertical (y) de la cara detectada en la imagen. El mensaje es de tipo *face_detector::ArrayInt*, y la información es procesada por una función de callback llamada *callback*.
2) **Cálculo de Ángulos Deseados**: En la función callback, los valores de x e y recibidos se mapean a los rangos de movimiento deseados para los servos. Esto implica transformar las coordenadas de la cara detectada (que van de 1 a 100) en ángulos específicos para los servos que controlan los iris y el cuello del robot. Por ejemplo, si la coordenada x indica que la cara está a la derecha, el sistema ajustará el ángulo del servo del iris para mirar hacia esa dirección y el ángulo del servo del cuello para orientar la cabeza en esa dirección.
3) **Control PID**: Para ajustar con precisión los servos, se utiliza un controlador PID. Este controlador compara la posición deseada con la posición actual de los servos y ajusta el movimiento para minimizar la diferencia. Hay tres controladores PID distintos en el código:

- **controlador_mirada**: Ajusta el ángulo de los iris para alinear la vista con la posición de la cara.
- **controlador_cuello**: Ajusta el ángulo del cuello para orientar la cabeza en la dirección de la cara.
- **controlador_cuello_abajo**: Ajusta el ángulo del cuello en dirección vertical para alinear la cabeza con la posición de la cara.

4) **Actualización de Posiciones**: Después de calcular los ángulos deseados y ajustar los controles PID, el sistema actualiza la posición de los servos. La función *cara.look_to()* mueve los iris hacia la posición deseada, mientras que *cara.raise_neck()* y *cara.turn_neck()* ajustan la orientación del cuello. Estos métodos se aseguran de que los componentes faciales sigan la posición de la cara detectada.

Además de seguir la cara, el robot realiza un parpadeo automático. Esto se gestiona en la función loop(), donde se activa un parpadeo aleatorio cada pocos segundos.

Levantar el programa seguidor de caras

El archivo launch *arduino_face_detector.launch* levanta los nodos de la cámara, al detector de caras y al que se encarga de la comunicación entre Arduino y ROS. Para levantarlo se debe ejecutar:

```
roslaunch face_detector arduino_face_detector.launch
```