INSTITUTO TECNOLÓGICO DE COSTA RICA

LABORATORIO LuTEC

Manual de arquitectura de robot KROTIC

Índice

1. Distribución de pines

Esta sección describe la distribución de pines en cada una de las tarjetas que se utilizan en la arquitectura de una forma gráfica y más sencilla que lo observado en el esquemático.

1.1. Distribución de pines nivel inferior

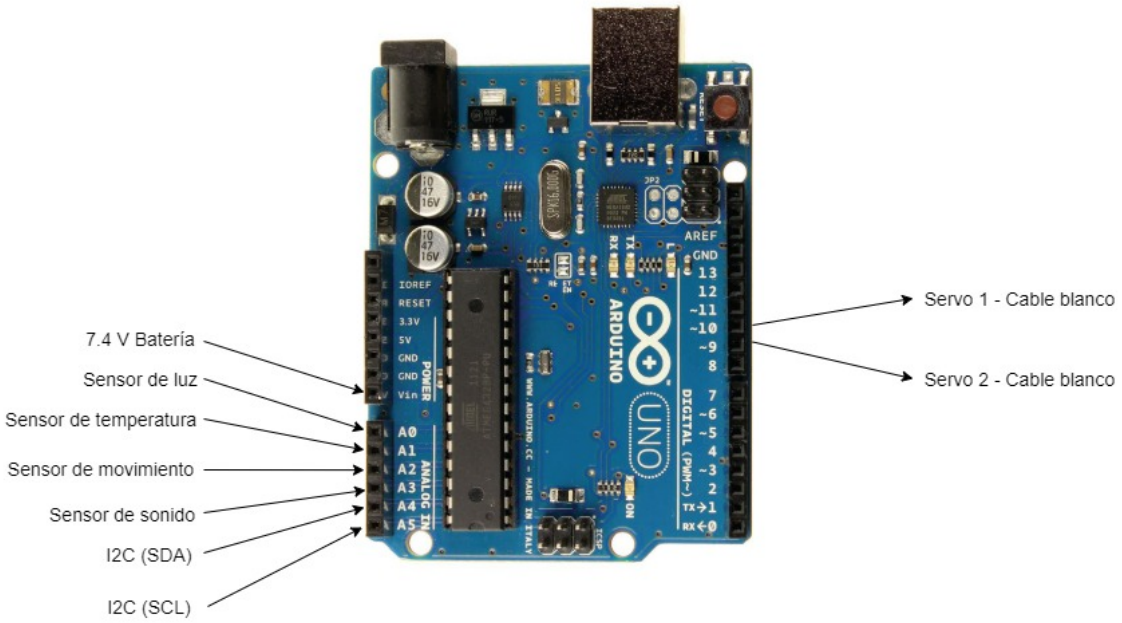

Figura 1: Distribución de pines nivel inferior.

Se tiene que, las entradas del ADC de la 0 a la 3 están ocupadas por los sensores del robot. Importante recordar que el Arduino no puede recibir en ninguno de sus pines (a excepción de la entrada Vin), un voltaje mayor a 5V, por lo que so alguno de los sensores supera este valor debe ser regulado.

Si se fuera a reemplazar la batería, recordar que el Arduino admite en su pin Vin valores entre 6 - 20 V. Sin embargo, lo recomendable es usar valores entre 7 - 12 V. Si se baja de eso puede provocar inestabilidad en la tarjeta y si se supera puede sobre calentar el regulador y dañarlo eventualmente.

Finalmente, Los pines A4 y A5 son reservados para la comunicación I^2C y los pines digitales 9 y 10 son para enviar señales a los servos.

1.2. Distribución de pines nivel superior

El nivel superior, que es manejado por una Raspberry Pi, como la mostrada en la figura 2. En su mayoría se encargar de controlar LEDs que pueden ser los tres que los estudiantes pueden manejar con el código o los de control, que son indicadores del estado de la batería

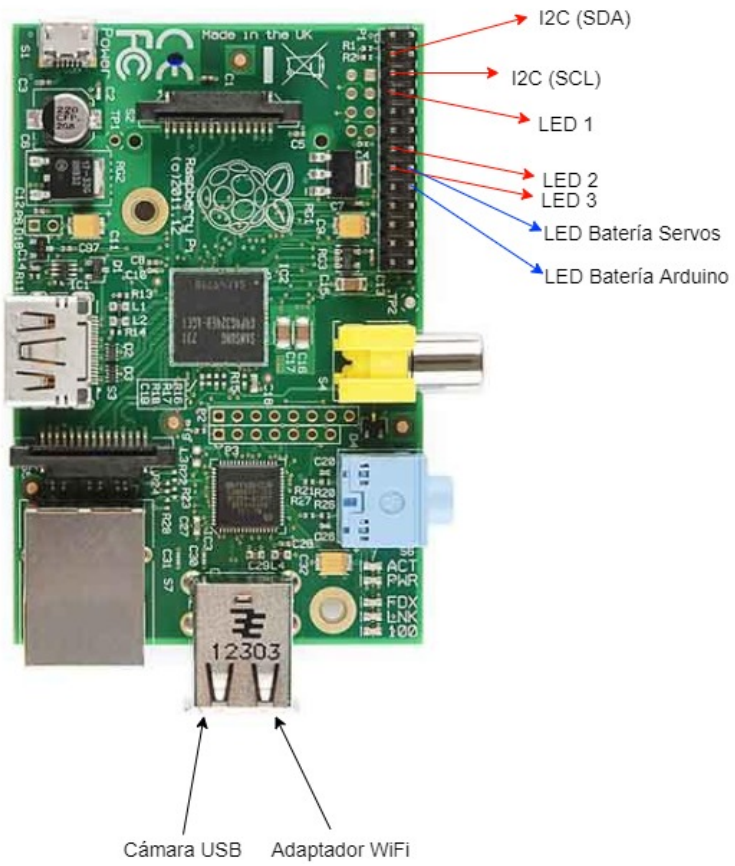

Figura 2: Distribución de pines nivel superior.

de los servos y del Arduino. Cuando el nivel de batería sobre pase cierto nivel los LEDs empezarán a parpadear.

Al igual que en el caso de Arduino, tener en cuenta la lógica que maneja la tarjeta, la Raspberry Pi maneja un nivel a 3.3 V, por lo que ninguno de sus pines debería recibir más de eso, porque se podría dañar la tarjeta. Por eso es que si se conectan dispositivos adicionales deberá regularse si es necesario.

En este nivel también se contempla, el uso de una cámara USB y el adaptador WiFi en ambos puertos USB.

1.3. Referencia de pines

Para futura referencia y para el código de prueba de python, tomar como referencia la siguiente figura, que contiene los pines con referencia al número en la tarjeta y las referencias según están definidos en el procesador. En el código de prueba se definen los pines como referencia a la tarjeta, y esto en el código se observa en el comando `GPIO.setmode(GPIO.BOARD)`, para esta configuración se utilizan los números que están al lado del nombre del pin en la figura 3, si se elige la otra referencia se tendría que especificar como `GPIO.setmode(GPIO.BCM)` y los números que se ponen son los números del GPIO.

3.3V	1	2	5V
I2C1 SDA	3	4	5V
I2C1 SCL	5	6	GROUND
GPIO4	7	8	UART TXD
GROUND	9	10	UART RXD
GPIO 17	11	12	GPIO 18
GPIO 27	13	14	GROUND
GPIO 22	15	16	GPIO 23
3.3V	17	18	GPIO 24
GPIO 10 MOSI	19	20	GROUND
GPIO 9 MISO	21	22	GPIO 25
GPIO 11 SCLK	23	24	GPIO 8
GROUND	25	26	GPIO 7

Figura 3: Distribución de pines Raspberry Pi.

2. Comunicación I2C

La mayor consideración que hay que tener en mente es que las dos tarjetas trabajan a niveles lógicos distintos, como ya se mencionó el Arduino funciona con 5V y la Raspberry a 3.3V. En este caso la comunicación I^2C se puede realizar conectando los cables de SDA y SCL directamente en cada tarjeta sin ningún problema porque la Raspberry es el maestro, y este es el que define el nivel lógico del bus. Entonces el arduino estará recibiendo alrededor de 3.3V en sus pines cuando exista comunicación, lo cual no es problema porque tiene un límite de 5V.

En caso de que se necesite cambiar esta configuración; es decir, el Arduino es maestro y la Raspberry esclavo, no se puede realizar una conexión directa porque la Raspberry estaría recibiendo más de 3.3V en sus pines y por lo tanto, dañándola. Si se quisiera hacer esta configuración, es necesario un componente extra, un convertidor lógico bidireccional como el de la figura 4 para poder bajar el nivel de 5V a 3.3V.

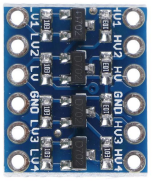

Figura 4: Convertidor lógico bidireccional.

2.1. Instrucciones I^{2}C

Las instrucciones que se envían desde la Raspberry al Arduino están codificadas conforme la siguiente tabla.

Cuadro 1: Instrucciones para el protocolo I^{2}C

Instrucción	**Variable en código Arduino**	**Valor I^{2}C**
Leer sensor de luz.	`READ_LIGHT_INSTR`	0x0
Leer sensor de temperatura.	`READ_TEMP_INSTR`	0x1
Leer sensor de movimiento.	`READ_MOTION_INSTR`	0x2
Leer sensor de sonido.	`READ_SOUND_INSTR`	0x3
Mover hacia adelante.	`MOVE_FORWARD`	0x4
Mover hacia atrás.	`MOVE_BACKWARDS`	0x5
Rotar hacia la derecha.	`ROTATE_RIGHT`	0x6
Rotar hacia la izquierda.	`ROTATE_LEFT`	0x7

3. Medición de temperatura

Como parte de los mecanismos de protección, se tiene que se puede monitorear la temperatura de la Raspberry Pi, con el objetivo de que si llegase a pasar cierto umbral que se considere alto, el dispositivo se apague para evitar daños. En este caso se ha definido un valor peligroso de temperatura para cualquier temperatura mayor a 70 ℃. En el código del programa de prueba (`KROTIC.py`) en la opción 11, se tiene el código necesario para realizar el monitoreo de temperatura y como apagar la tarjeta si se pasa el umbral.

El comando que mide la temperatura es `sudo vcgencmd measure_temp` y en el código luego se toma la parte de la respuesta que interesa utilizando indexación de *strings* y luego si el valor medido es más alto que el umbral mencionado se ejecuta el comando `sudo shutdown -h now` que apagará la Raspberry inmediatamente.

4. Transmisión en vivo

La transmisión en vivo se hace a través de la biblioteca motion, que se puede instalar con el comando `sudo apt-get install motion`, la cual monta un servidor de transmisión como proceso que corre como *daemon*. Ya el servidor se ha configurado, entonces los pasos para levantar el servicio son:

1. Conectarse a internet.
2. Abrir la terminal y escribir el comando `hostname -I`, esto le dará la IP de la Raspberry.
3. Para activar el servidor escribir en la terminal `sudo service motion restart` o `sudo service motion start`.
4. Una vez finalice el proceso del comando anterior, escribir en la terminal `sudo motion`.

5. Para verificar que se haya levantado el servidor entrar al navegadory escribir: `<Raspberry IP>:8081`. En `<Raspberry IP>` poner la salida del comando del paso 2 y el puerto por defecto es el 8081. Al entrar se visualizará lo que esté capturando la cámara.

Si se necesitara cambiar la configuración, referirse al siguiente instructivo, donde explican como levantar el servicio desde cero y donde encontrar los archivos de configuración.

Ahora, por el momento se realiza un streaming que solo es accesible por la red local. Para convertirlo en un servicio que pueda ser accesado en cualquier parte se debe hacer el proceso de redireccionamiento de puertos o *port forwarding*.

El procedimiento cambia de acuerdo al router que se tenga, por lo que lo más recomendable es contactar a la proveedora de internet para que se asista en el proceso.

5. Manejo de cámara

Si se quiere realizar pruebas con la cámara independientemente de la transmisión en vivo, instalar la biblioteca `fswebcam` y `ffmpeg` con el comando `sudo apt install fswebcam` y con `sudo apt-get install ffmpeg`

1. Para tomar fotos y guardarlas en disco usar el comando `fswebcam image.jpg`. Después del comando `fswebcam` debe ir el *path* donde se quiere guardar la foto junto con el nombre de la foto y su formato, si se quiere guardar en el mismo directorio poner solo el nombre y el formato.

2. Para grabar un video y guardarlo en disco utilizar el comando `ffmpeg -f v4l2 -i /dev/video0 -t X output.mp4`, donde `X` es el valor en segundos de la duración del video.

6. Manejo de entradas analógicas en la Raspberry Pi

Como la Raspberry Pi es la encarga de manejar la protección del sistema, debe medir el nivel de voltaje de las baterías para avisar cuando estas se encuentren cerca de descargarse. Esto implica que se deben tomar los valores analógicos de voltaje de las baterías y que la Raspberry Pi los lea. Los pines de la Raspberry son todos digitales, por lo que se necesita un ADC para poder leer los valores de las baterías correctamente. En el esquemático del diseño, se utiliza un MCP3008 que es bastante común y se tiene mucha información en diferentes fuentes sobre como manejar el dispositivo.

A grandes rasgos, el ADC es un dispositivo que se comunica con la Raspberry a través del protocolo SPI. Además tiene soporte para leer varios dispositivos al mismo tiempo, específicamente, tiene 8 canales para lectura de datos al mismo tiempo. Los pines no se pusieron en la figura 2 porque puede ser que se utilice otro ADC distinto, entonces eso quedará para implementación futura. Sin embargo, para leer los valores desde python, se puede utilizar el siguiente código:

```
import spidev
import time
```

```
#Define Variables
delay = 0.5
battery_servo_channel = 0
battery_arduino_channel = 1

#Create SPI
spi = spidev.SpiDev()
spi.open(0, 0)

def read_adc(channel):
    # read SPI data from the MCP3008, 8 channels in total
    if channel > 7 or channel < 0:
        return -1
    r = spi.xfer2([1, 8 + channel << 4, 0])
    data = ((r[1] & 3) << 8) + r[2]
    return data


while True:
    servo_value = read_adc(battery_servo_channel)
    arduino_value = read_adc(battery_arduino_channel)

    print("----------------------------------------")
    print("Servo battery level: %d" % servo_value)
    print("arduino battery level: %d" % arduino_value)
    print("----------------------------------------")
    time.sleep(delay)
```

La biblioteca externa que se utiliza es spidev y se puede instalar a través de pip, con el comando `pip install spidev`.

7. Consumo y fuentes de alimentación

En cuanto al consumo de energía, la Rasberry Pi es el componente que más consumo tiene, esto porque si se considera como base la corriente que necesita cuando se está activa sin ningún periférico conectado, se tienen inicialmente 500 mA, luego se tienen 5 leds indicadores. El consumo de esos LEDs se calcula como:

$$I_{led} = \frac{V_{rpi} - V_{led}}{R_{led}} \tag{1}$$

Donde, V_{rpi} es la salida de voltaje de la Raspberry, que es 3.3V, V_{led} es la caída de voltaje del LED, que según la página donde se hizo la compra tienen una caída de 2.2V y R_{led} es una resistencia de 220 Ω. Si se hace el cálculo, cada LED necesita 5 mA, y como son cinco, los LEDs abarcan 25 mA.

Luego, los periféricos que más corriente necesitan son la cámara y el adaptador WiFi, que pueden llegar a un máximo de 600 mA cada uno.

Si se suman estos valores tenemos que la corriente necesario solo para el nivel superior es de $I_s = 600 \cdot 2 + 25 + 500$ mA, se tiene que I_s = 1.7 A.

Por lo que recomendable es utilizar una fuente que provea de al menos 2 A constantes para asegurarse de que todos los servicios funcionan bien.

8. Regulación de voltaje

Según el esquemático de la arquitectura, para las entradas de los voltajes que indican el nivel de batería, se utilizaron divisores de voltaje para regular la entrada a los pines de la tarjeta. Recordar que la Raspberry, que es la que maneja todos los LEDs indicadores, soporta no más de 3.3V ens su pines.

Entonces, para el caso de las baterías de los servos. Se usan 4 baterías AA, por lo que cuando están nuevas pueden llegar a dar alrededor de 6.4 V. El divisor de voltaje para este caso se da con dos resistencias; una de 100kΩ y otra de 107kΩ, el voltaje de entrada para la Raspberry se toma desde la resistencia de 107kΩ que al aplicar el divisor se obtiene:

$$V_{in} = \frac{107 \cdot 6{,}4}{100 + 107} = 3{,}3V. \tag{2}$$

Coincidiendo este valor máximo de carga con el valor de referencia del ADC.

Para el caso de la batería del Arduino, en su máxima carga tendrá una salida aproximada de 7.4 V, los valores de resistencia son de 100kΩ y otra cercana al 80.5kΩ tomando la entrada al ADC desde esta última resistencia de modo que al aplicar el divisor:

$$V_{in} = \frac{80{,}5 \cdot 7{,}4}{100 + 80{,}5} = 3{,}3V. \tag{3}$$

Para que se tenga que la tensión en la carga máxima coincide con el voltaje de referencia del ADC.