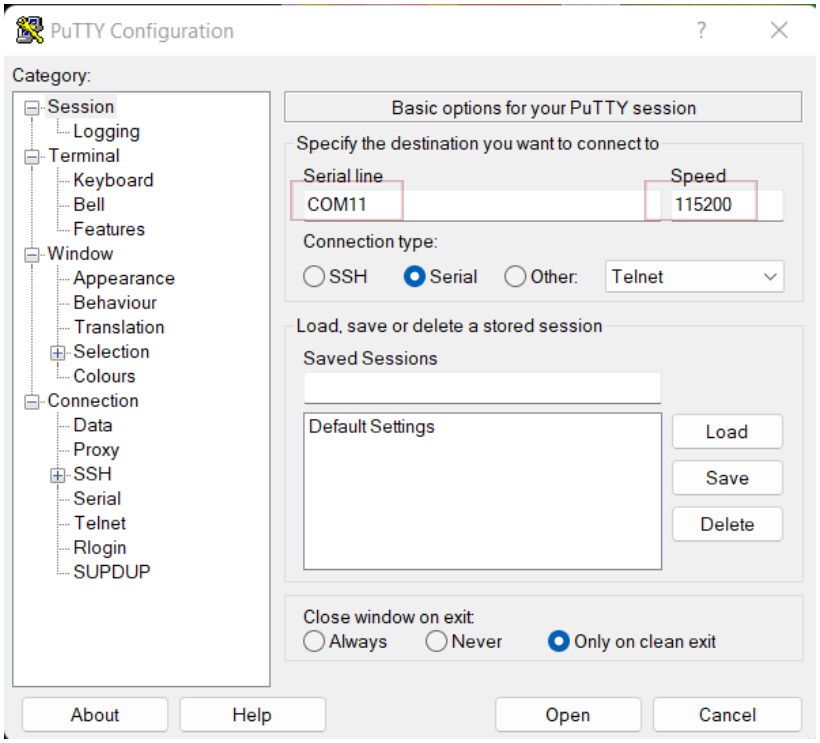
About Coral Dev Board

反對無效

All my work on coral can be found at
https://github.com/KyawSawLIN/PTZ-coral-backup.git

See https://coral.ai/docs/dev-board/get-started/ for reference.
There are three ways to control and program the coral board. Username is mendel and password is also mendel.

1. First is to connect via the mdt shell. See details at https://coral.ai/docs/dev-board/get-started/#connect-via-mdt (see pdf in github)
2. Second is using Putty via serial communication with your computer. See details at https://coral.ai/docs/dev-board/serial-console/#connect-with-windows (see pdf in github)



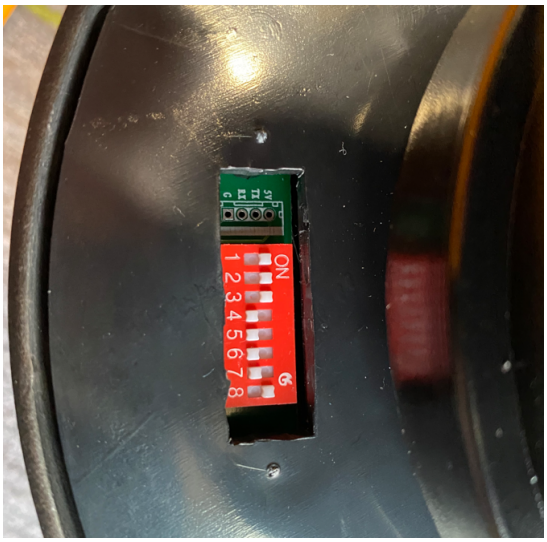
3. Third is using SSH. To use ssh, you first need to know the coral IP address. So you need to apply one of the first two ways to get into the terminal of coral linux.That step is needed to type "nmcli device show" command in coral shell to get the IP address. Then, in your git bash terminal (if windows) or in mac terminal (ssh is built-in in mac), type ssh mendel@coral-ip-address.

I mostly log in as a root user. To change from user mendel to root user, type sudo-i. mendel@fun-pig will change to root@fun-pig. See NOTES_by_KSL.txt and tbd folder inside root. All python programs must be run on python3 and in a virtual environment. If you clone this "git clone https://github.com/KyawSawLIN/PTZ-coral-backup.git" repository, all the necessary packages are already included and type ". bin/activate" to activate the virtual environment.

Controlling PTS303 platform

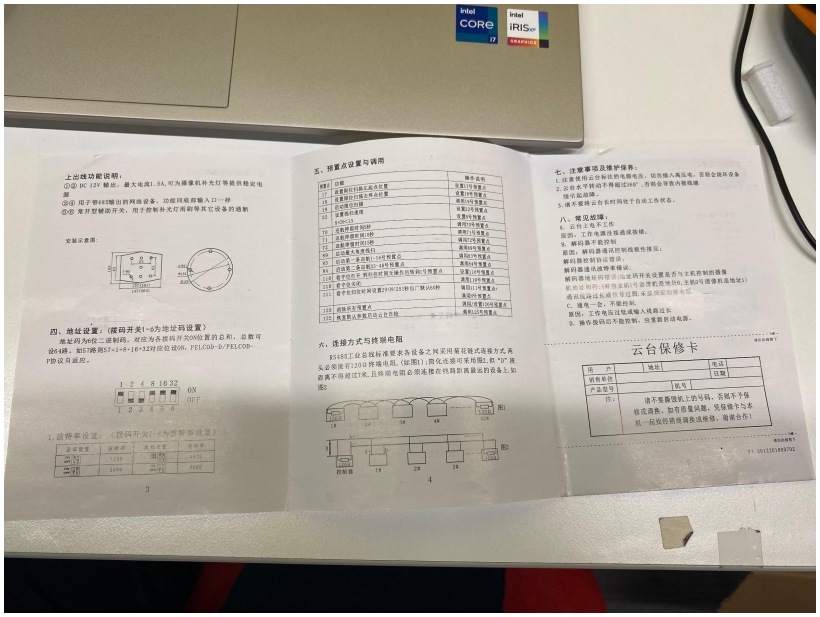
About Hardware requirements

PTZ in this project is PTS-303Z. (12V DC input). This is a pan(horizontal) and tilt(vertical) moving platform. Inputs are rs485+, rs485-, 5V and ground. You need to make sure to check there is a connected port and replace the name if necessary. At the end of the PTS 303Z platform, there is an adjustable baud rate where you can change depending on your need.

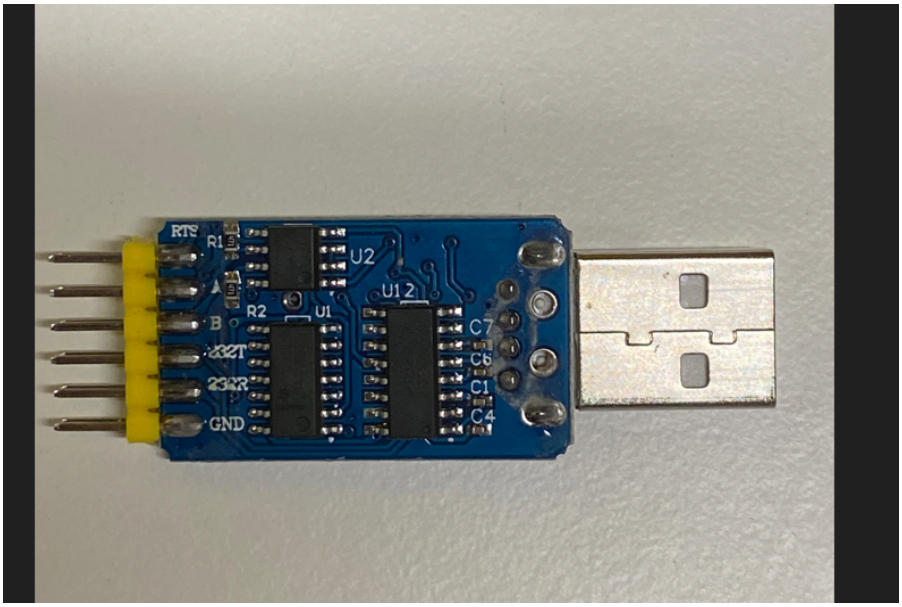


In the above photo, switches from 1 to 6 are for naming pan-tilt platforms (as these platforms are used not alone but in mass deployment) and it is counted in binary. In this photo, the number is 7 which should be inputted to the 2nd bit of pelco-d format. You can choose any number you like by changing 1-6 switches and of course change it in software correspondingly. See “07” in the photo below. 7 and 8 switches are for baud rate control.

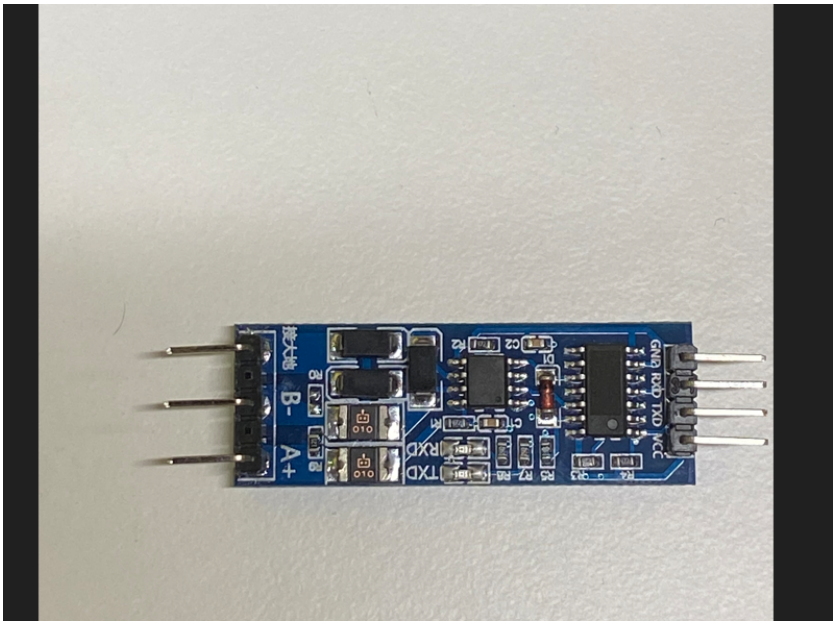
```
ser.write(bytes.fromhex('FF 07 00 51 00 00 58'))
```



I use coral dev board UART pins to generate serial data. You can also use serial ports in PC(for PC, you may need a usb2uart-rs485-rs323 converter see photo in github) or any microcontroller that includes UART.



As this PTZ platform works via RS485 serial hardware. You need a uart2rs485 see photo in github converter(MAX485 chip) to control.



So, the procedure is the program generates serial data which is in uart and we change it to rs485 and feed it into the ptz platform.

Chip, line	Device path	Pin function	Pin		Pin function	Device path	Chip, line
		+3.3 V	1	2	+5 V		
	/dev/i2c-1	I2C2_SDA	3	4	+5 V		
	/dev/i2c-1	I2C2_SCL	5	6	Ground		
	/dev/ttymxc2	UART3_TXD	7	8	UART1_TXD	/dev/ttymxc0	
		Ground	9	10	UART1_RXD	/dev/ttymxc0	
	/dev/ttymxc2	UART3_RXD	11	12	SAI1_TXC		
0, 6	/sys/class/gpio/gpio6	GPIO_P13	13	14	Ground		
2, 0	/sys/class/pwm/pwmchip2/pwm0	PWM3	15	16	GPIO_P16	/sys/class/gpio/gpio73	2, 9
		+3.3 V	17	18	GPIO_P18	/sys/class/gpio/gpio138	4, 10
	/dev/spidev0	ECSPI1_MOSI	19	20	Ground		
	/dev/spidev0	ECSPI1_MISO	21	22	GPIO_P22	/sys/class/gpio/gpio140	4, 12
	/dev/spidev0	ECSPI1_SCLK	23	24	ECSPI1_SS0	/dev/spidev0.0	
		Ground	25	26	ECSPI1_SS1	/dev/spidev0.1	
	/dev/i2c-2	I2C3_SDA	27	28	I2C3_SCL	/dev/i2c-2	
0, 7	/sys/class/gpio/gpio7	GPIO_P29	29	30	Ground		
0, 8	/sys/class/gpio/gpio8	GPIO_P31	31	32	PWM1	/sys/class/pwm/pwmchip0/pwm0	0, 0
1, 0	/sys/class/pwm/pwmchip1/pwm0	PWM2	33	34	Ground		
		SSAI1_TXFS	35	36	GPIO_P36	/sys/class/gpio/gpio141	4, 13
2, 13	/sys/class/gpio/gpio77	GPIO_P37	37	38	SAI1_RXD0		
		Ground	39	40	SAI1_TXD0		

You can use above uart pins but I use USB output from coral as it is less error-prone. Type "python3 -m serial.tools.list_ports" to see any connected USB ports and for me it is **"/dev/ttyUSB0"**

About Software

➔ I use pyserial to generate serial data by python. Regarding pyserial API, please refer to https://pythonhosted.org/pyserial/pyserial_api.html.

➔ The control variables here are **port** and **baud rate**. One thing to note is that in order to control the PTZ platform, the serial data should be in pelco-d format which is in 7 bytes hexadecimal [see pelco-d pdf in github].

Depending on what you want to do, you can send commands (eg.. Read the current position, make it go to 180 degree) , in pelco-d format and PTZ will respond accordingly.

The program will run on an infinite loop and if you want to kill the program, use keyboard_interrupt Ctrl+C.

The name of the code is test.py [see test.py in github].

Behavior : user will be asked to choose pan or tilt and can enter pan degree(0-360) , tilt(0-180).

For patrolling pan program,
Step-by-step instructions,

- First connect uart2rs485 [see photo in github] to usb port in any board that you want to work (coral, jetson, rk3588).
- Then, connect tx and rx of uart2rs485 [see photo in github] to tx and rx of uart2rs485 [see photo in github] which inturn connects to the linker (rs485+ and rs485- of PTS303Z.
- Type “python3 -m serial.tools.list_ports” to see connected usb ports.
-
```
(final) firefly@firefly:~/final$ python3 -m serial.tools.list_ports
/dev/ttyUSB0
1 ports found
(final) firefly@firefly:~/final$
```
- Then go to the folder that you want to work and type “python3 -m venv .\”. Type “ls” and it will be like this.
-
```
firefly@firefly:~$ mkdir and cd kkkk
firefly@firefly:~$ cd kkkk
firefly@firefly:~/kkkk$ python3 -m venv .\
>
firefly@firefly:~/kkkk$ ls
bin  include  lib  lib64  pyvenv.cfg  share
```

- Type ". bin/activate" to activate the virtual environment.
- Then install the pyserial package in the created virtual environment with "pip install pyserial".
- Copy patrol.py in github. You can skip the following if you know how to use vim. First,type "vim patrol.py" to create a python file and press "i" to insert, then paste(right click) what you copied before. Then, press "esc" and "ctrl+w" and "ctrl+x". Then, the file will be saved and you can run with "python3 patrol.py".
- If you want to repeat pan patrolling from 0 to 360 degree, modify the array below. Also modify the special counter to respective value in func function.
-

```
83    f = [45,90,135,180,135,90,45,0]  # repeating pan angles
```

```
def func(special_counter = 0 ):
    while 1:
        tmp = mod(f[special_counter])
        var_0 = toHex(tmp)
        pan(var_0)
        time.sleep(10) # replace any time you want the pan patrolling to stop for instead of 10s
        special_counter += 1
        if special_counter == 8: # reset f to start bit
            special_counter = 0
        print('Thread alive, but it will die on program termination')
```

Controlling 2 individual servos - pan and tilt

About Hardware requirements

➔ Both servos need 4.8 to 8 V. Three input wires are 8V, ground and control (PWM). I use PWM pins of coral to generate PWM. Operating frequency is 50Hz and so full pulse width is 20ms.

➔ The working pulse width is 0.5 ms to 2 ms. You have to generate pulse width between 0.5ms to 2 ms to get to the angle that you want.

About software

Pan angle is from 0 to 180 while tilt angle ranges from 0 to 180.

Pan angle rotates in linear incremental manner while I program tilt motion in non-linear behavior to smooth the movement respective of the load.

Normally, servo runs to a specific angle according to PWM instantly. But the tilt servo doesn't work that way and I had to implement to rotate it 5 degrees at a time. Doing so also gives me an advantage of letting me control the servo speed. To do that, I need to know the previous angle in order to rotate to the user-input angle by subtracting or adding to that previous angle. If the code is running for the first time, the default previous tilt angle is 90 and pan is 0. Here I implement pan motion in linear increment and tilt in non-linear increment which makes tilt motion similar to damped motion.

See the code and details at sp.py [see sp.py in github].

Behavior : pan servo will be patrolling around 0-180 (45 degree at a time and stop there for a while), user can input any tilt degree (0-180) at the same time.

How to run

1. git clone https://github.com/KyawSawLIN/PTZ-coral-backup.git

2. Type "cd PTZ-coral-backup" and type "pip install pyserial"
3. type ". bin/activate" to activate virtual env (all the necessary packages are already included)

4. Then run python3 sp.py for 2 servos control and python3 test.py for the PTS303Z platform.

About Jetson Board and RK3588

Controlling PTS303 platform

All my work on jetson nano can be found at https://github.com/KyawSawLIN/PTZ-jetson.

Controlling the PTS303 platform in jetson nano is the same as it is in google coral. You just need to make sure there is pyserial downloaded in your package. If not, try "pip install pyserial" and activate the virtual env.

You will need to reconfigure the port for both Jetson nano and RK 3588 as it won't be the same with coral.

```
# You should be familiar with pelco-d format to understand this


import serial
import time

ser = serial.Serial("/dev/ttymxc2", 2400)  # 2400 baud rate
```

For jetson nano

The Hard Way - Using Built-in UART

If you don't want to plug in external hardware to the Jetson Nano you can use the built in UART on the RX/TX pins. Unlike the Raspberry Pi, the Jetson Nano isn't using the RX/TX pins for a console, those are on a different UART peripheral, so you should be good to go!

You can use the built in UART via `/dev/ttyTHS1`

Wire the GPS as follows:

- Connect the Jetson Nano +3.3V pin to the Vin pin on the GPS
- Connect the Jetson Nano Ground pin to the GPS Ground pin
- Connect the Jetson Nano UART TX (#8) to the GPS RX pin
- Connect the Jetson Nano UART RX (#10) to the GPS TX pin

Jetson Nano Dev-Board Expansion Header

Alt Function	Linux(BCM)	Board Label			Board Label	Linux(BCM)	Alt Function
DAP4_DOUT	78(21)	D21	40	39	GND		
DAP4_DIN	77(20)	D20	38	37	D26	12(26)	SPI2_MOSI
UART2_CTS	51(16)	D16	36	35	D19	76(19)	DAP4_FS
		GND	34	33	D13	38(13)	GPIO_PE6
LCD_BL_PWM	168(12)	D12	32	31	D6	200(6)	GPIO_PZ0
		GND	30	29	D5	149(5)	CAM_AF_EN
		D1/ID_SC	28	27	D0/ID_SD		
SPI1_CS1	20(7)	D7	26	25	GND		
SPI1_CS0	19(8)	D8	24	23	D11	18(11)	SPI1_SCK
SPI2_MISO	13(25)	D25	22	21	D9	17(9)	SPI1_MISO
		GND	20	19	D10	16(10)	SPI1_MOSI
SPI2_CS0	15(24)	D24	18	17	3.3V		
SPI2_CS1	232(23)	D23	16	15	D22	194(22)	LCD_TE
		GND	14	13	D27	14(27)	SPI2_SCK
DAP4_SCLK	79(18)	D18	12	11	D17	50(17)	UART2_RTS
		RXD/D15	10	9	GND		
		TXD/D14	8	7	D4	216(4)	AUDIO_MCLK
		GND	6	5	SCL/D3		
		5V	4	3	SDA/D2		
		5V	2	1	3.3V		

replace “/dev/ttymxc2” with “/dev/ttyTHS1” in test.py [see test.py in github]().

For RK3588

8.2. DTS Configuration

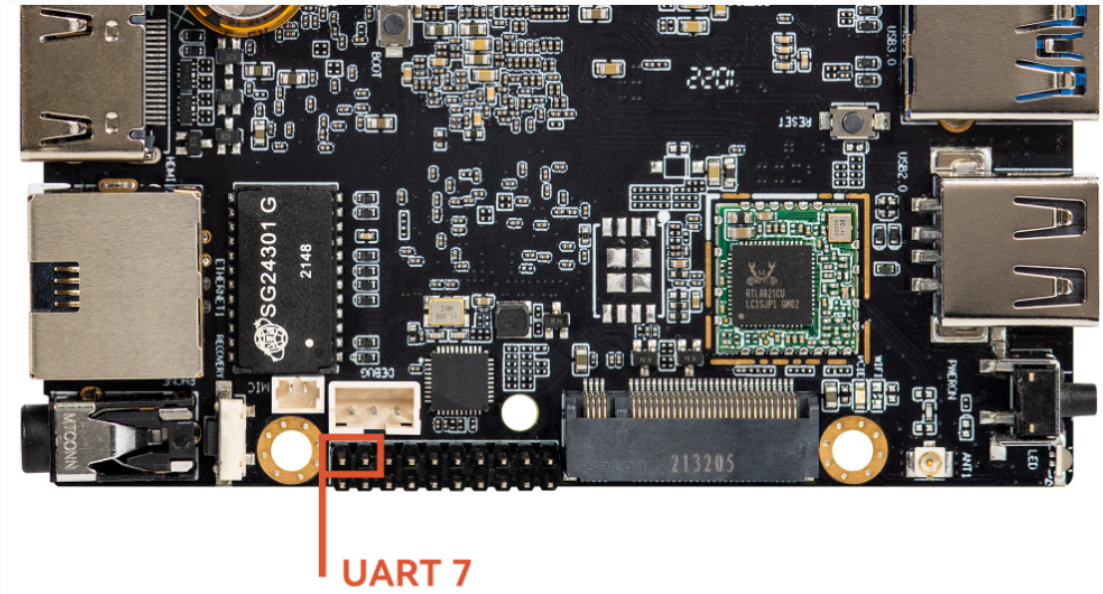
file path `kernel-5.10/arch/arm64/boot/dts/rockchip/roc-rk3588s-pc.dtsi`

```
/* uart7 */
&uart7{
    pinctrl-0 = <&uart7m2_xfer>;
    status = "okay";
};
```

After configuring the serial port, the nodes on the software corresponding to the hardware interface are:

```
UART7:    /dev/ttyS7
```

The serial port interface diagram of ROC-RK3588S-PC hardware version is as follows:



replace "/dev/ttymxc2" with "/dev/ttyS7" in test.py [see test.py in github](#).

For more details on RK3588 reference,
https://wiki.t-firefly.com/zh_CN/ROC-RK3588S-PC/usage_uart.html

For more details on jetson reference,
https://github.com/KyawSawLIN/PTZ-jetson/blob/main/circuitpython-libraries-on-linux-and-the-nvidia-jetson-nano.pdf.

For more details on google coral reference,
https://github.com/KyawSawLIN/PTZ-coral-backup/blob/main/Coral-getting-started.pdf
https://github.com/KyawSawLIN/PTZ-coral-backup/blob/main/Coral-IO-connect.pdf