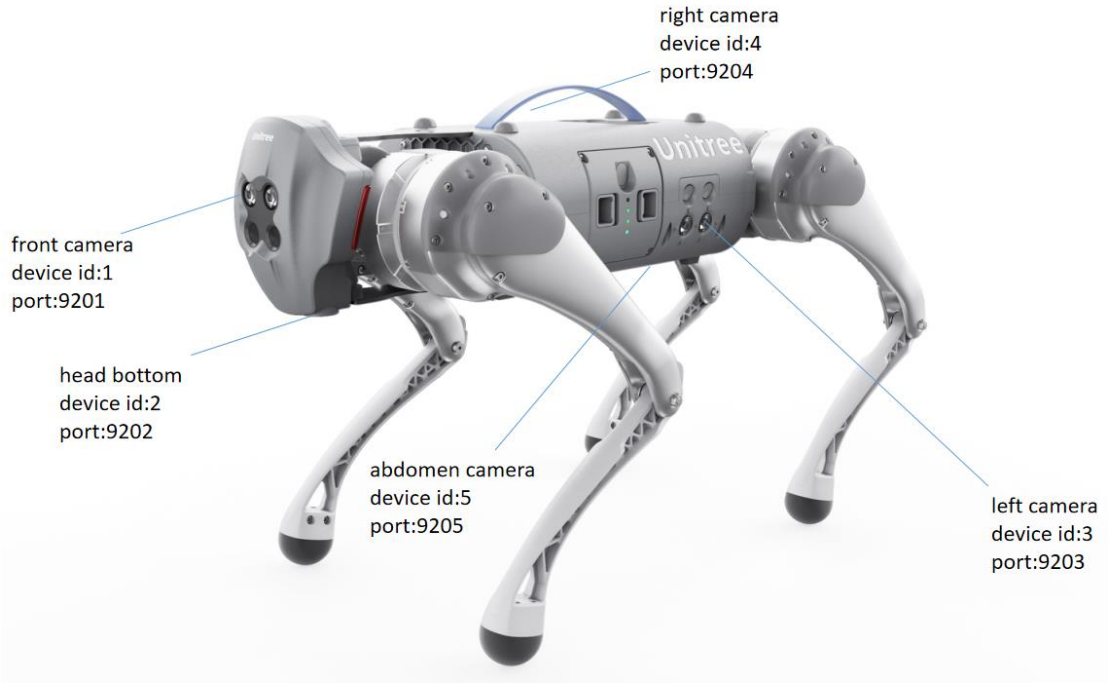
Development and use of Go1 binocular fisheye camera

Introduction

The head and body of Go1-Edu are distributed with 5 sets of binocular fisheye cameras, and we provide the corresponding UnitreeCameraSDK for developers to use.

1. hardware

Five groups of cameras are distributed in the front face, chin, left and right sides of the fuselage and abdomen. Hardware, the front face camera and chin camera are connected to the head Nano (192.168.123.13), the cameras on both sides of the body are connected to the body Nano (192.168.123.14), and the abdomen camera is connected to the main body Nano or NX (192.168.123.15). The schematic diagram is as follows.



In the system of each motherboard, device IDs (dev IDs) are assigned to the cameras:

- Head Nano (192.168.123.13), front-facing camera dev ID=1, chin camera dev ID=0.
- Body Nano (192.168.123.14), left camera dev ID=0, right camera dev ID=1.
- Body main Nano (192.168.123.15), abdominal camera dev ID=0.

For the convenience of the receiving program, we define ports (Port ID) for each camera as the table shown below:

PORTs for cameras:

Nano	dev ID	Port ID	Position
Nano（192.168.123.13）	1	9201	Front
Nano（192.168.123.13）	2	9202	Head bottom（chin）
Nano（192.168.123.14）	3	9203	left
Nano（192.168.123.14）	4	9204	right
Nano（192.168.123.15）	5	9205	abdomen

Camera hardware parameters:

RGB	RGB frame resolution:	1856x800 (stereo)
	RGB frame rate:	30fps
	RGB sensor technology:	Global Shutter
	RGB sensor FOV:	222°
Depth	Depth technology:	Stereoscopic
	Effective depth range:	10cm~85cm
	Depth Field of View:	＜180°
	Depth output resolution:	adjustable (default464x400)
	Depth frame rate:	≤30fps

2. SDK

The camera SDK is listed on the Yushu GitHub repository, so you can download it and use it yourself.
https://github.com/unitreerobotics/UnitreecameraSDK

The camera SDK provides video streams in color and depth, as well as internal calibration information, and also provides point clouds and depth images aligned with the color images.

The SDK includes the following:

example_getCalibParamsFile.cc -- Get camera calibration parameters

example_getDepthFrame.cc -- Get the camera depth map

example_getPointCloud.cc -- Get the camera point cloud

example_getRawFrame.cc -- Get the camera's unprocessed image

example_getRectFrame.cc -- Get the processed image of the camera

example_getimagetrans.cc -- receive images from the network

example_putImagetrans.cc -- Send images over the network

example_share.cc -- Shared memory code routines, not open to users

Among them, the first five are local calls to the camera, which can be compiled and run directly; the two transmission samples, one sending and one receiving, need to be run separately.

3. SDK Usage examples

Before the SDK use, you need to kill the program that comes with the call to the camera in order to successfully obtain the image. There are three programs related to the camera, point_cloud_node, mqttControlNode, live_human_pose , respectively, point cloud image, transmission, human body recognition, you can use the ps -aux command to query the status, kill these processes before use to lift the camera occupation, release arithmetic.

```
ps -aux | grep point_cloud_node
ps -aux | grep mqttControlNode
ps -aux | grep live_human_pose
ps -aux | grep point_cloud_node | awk '{print $2}' | xargs kill -9
ps -aux | grep mqttControlNode | awk '{print $2}' | xargs kill -9
ps -aux | grep live_human_pose | awk '{print $2}' | xargs kill -9
```

For your understanding, let's take the example of getting the image of the dog's head camera on your own computer.

3.1 Acquire a single image from the camera in front of the head

3.1.1 Download

```
git clone https://github.com/unitreerobotics/UnitreecameraSDK.git
```

3.1.2 Connect your own PC to the dongle via cable and configure the network

The following is an example of configuring a local IP segment, please follow the actual situation.

```
sudo ifconfig eth0 192.168.123.100/24
sudo ifconfig eth0 up
ifconfig
```

3.1.3 Send camera sdk to head Nano

Since the header Nano does not provide HDMI and USB for customers to use, we can send the SDK folder to the header Nano via the scp tool.

```
scp -r UnitreecameraSDK unitree@192.168.123.13:/home/unitree/
```

3.1.4 Remote connection head Nano

```
ssh unitree@192.168.123.13
password:123
```

3.1.5 Stop the camera-related processes that come with the head Nano

```
ps -aux | grep point_cloud_node | awk '{print $2}' | xargs kill -9
ps -aux | grep mqttControlNode | awk '{print $2}' | xargs kill -9
```

As mentioned at the beginning of this section, we need to kill the point_cloud_node, mqttControlNode process (live_human_pose is only available on the main Nano of the body). After kill, you can use the query process command to confirm again if the process was successfully kill.

3.1.6 Modify the transfer configuration parameters file trans_rect_config.yaml

```
cd UnitreecameraSDK
vim trans_rect_config.yaml
%YAML:1.0
---
# [pls dont change] The log level
LogLevel: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 1. ]
# [pls dont change] The threshold is applied to detecd point cloud
Threshold: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 190. ]
#[pls dont change] it's a switch for a algorithm in the process of computing stereo disparity
Algorithm: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [1. ]
#UDP address for image transfer   192.168.123.IpLastSegment
IpLastSegment: !!opencv-matrix
```

```
   rows: 1
   cols: 1
   dt: d
   data: [ 100. ]
#DeviceNode
DeviceNode: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 1. ]
#fov (perspective 60~140)
hFov: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 90. ]
#image size ([1856,800] or [928,400])
FrameSize: !!opencv-matrix
   rows: 1
   cols: 2
   dt: d
   data: [ 1856., 800. ]
#rectified frame size
RectifyFrameSize: !!opencv-matrix
   rows: 1
   cols: 2
   dt: d
   data: [ 928., 800. ]
#FrameRate, it gives the limitation for transmission rate(FPS)
FrameRate: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
```

```
   data: [ 3e+01 ]
#0 ori img - right  1 ori img - stereo  2 rect img - right  3 rect img - stereo   -
1 不传图
Transmode: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 2. ]
#Transmission rate(FPS) in the UDP transmitting process. <= FrameRate
Transrate: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 3e+01 ]
# [pls dont change]  It's a switch in distortion process of fisheye camera. 1
represents "Longitude and latitude expansion of fisheye camera";  2 represnets
"Perspective distortion correction".
Depthmode: !!opencv-matrix
   rows: 1
   cols: 1
   dt: d
   data: [ 1. ]
 # empty reserved IO
Reserved: !!opencv-matrix
   rows: 3
   cols: 3
   dt: d
```

3.1.7 Compile and run example_putImagetrans

- Build

```
cd UnitreecameraSDK
mkdir build
cd build
cmake ..
make
```

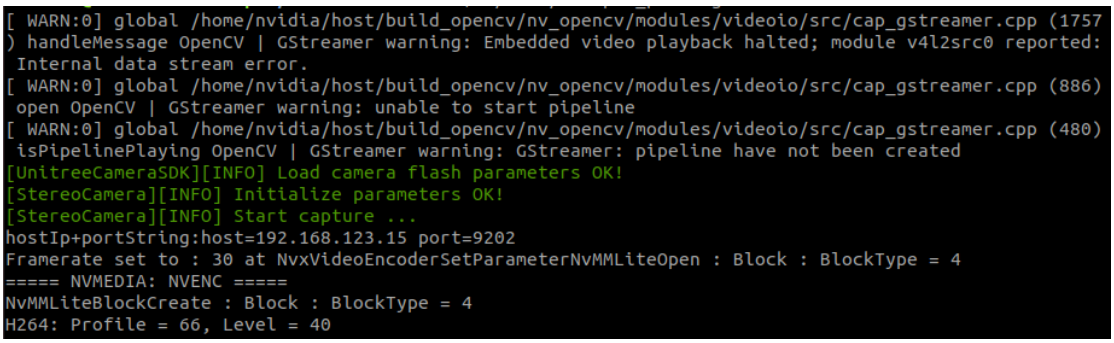
- Run

  Once compiled, the bins folder will be generated in the UnitreecameraSDK directory to run the sending program.

```
./bins/example_putImagetrans
```

Note: Do not run in cd to bins, this will be missing dependencies

Run successfully as shown below:



Note: When we recompile and run the sending program, sometimes it will report an error, which may be related to the sending process not kill clear.

In this case, you can kill the following program and try again.

ps -aux | grep `send_image_client` | awk '{print $2}' | xargs kill -9

3.1.8 Compile and run example_getimagetrans on your own PC

- example_getimagetrans.cc

```
#include <opencv2/opencv.hpp>
#include <iostream>
```

```
int main(int argc,char** argv)
{
    std::string IpLastSegment = "15";//Change to the IP of the current PC
    int cam = 1;//Front port is 1 (9201)
```

- Build

```
cd UnitreecameraSDK
mkdir build
cd build
cmake ..
make
```

- Run

```
./bins/examples_getimagetrans
```

The program runs normally and we can see the acquired image:

Note: If the PC is AMD64 (X86) platform, you need to modify the decoder in the receiver program:

```
std::string udpstrBehindData = " ! application/x-rtp,media=video,encoding-name=H264 ! rtph264depay ! h264parse ! avdec_h264 ! videoconvert ! appsink";
```

If the PC is an ARM64 platform, it is sufficient to keep the original decoder:

```
std::string udpstrBehindData = " ! application/x-rtp,media=video,encoding-name=H264 ! rtph264depay ! h264parse ! omxh264dec ! videoconvert ! appsink";
```

3.2 Simultaneous acquisition of images from the head front camera and the chin camera

Sometimes we need to get the screen of two cameras on a Nano at the same time, at this time we can create two transmission configuration

files trans_rect_config_cam1.yaml and trans_rect_config_cam2.yaml, and then create two transmission programs to run separately to send both cameras out at the same time. Of course, it is also possible to transmit twice, in the same program.

For example:

```
int main(int argc, char *argv[])
{
    UnitreeCamera cam0("trans_rect_config_cam1.yaml"); ///< init camera by device node number
    UnitreeCamera cam1("trans_rect_config_cam2.yaml"); ///< init camera by device node number
    if(!cam0.isOpened())   ///< get camera open state
        exit(EXIT_FAILURE);
    if(!cam1.isOpened())   ///< get camera open state
        exit(EXIT_FAILURE);
    cam0.startCapture(true,false); ///< disable share memory sharing and able image h264 encoding
    cam1.startCapture(true,false); ///< disable share memory sharing and able image h264 encoding

    usleep(500000);
    while(cam0.isOpened()&&cam1.isOpened())
    {
        cv::Mat left0,right0, left1, right1;
        if(!cam0.getRectStereoFrame(left0,right0))
        {
            usleep(500);
            continue;
        }
         if(!cam1.getRectStereoFrame(left1,right1))
        {
            usleep(500);
```

```
            continue;
        }
        char key = cv::waitKey(10);
        if(key == 27) // press ESC key
           break;
    }

    cam0.stopCapture(); ///< stop camera capturing
    cam1.stopCapture(); ///< stop camera capturing
    return 0;
}
```

4. Write image receiving program using python

```
# -*- coding: utf-8 -*-
'''copyright Copyright (c) 2020-2021, Hangzhou Yushu Technology Stock CO.LTD. All
Rights Reserved
writen by Zhentao Xie
supporter: Zhentao Xie xiezhentao_1998@163.com
'''

import cv2

class camera:
    def __init__(self, cam_id = None, width = 640, height = 480):
        self.width = 640
        self.cam_id = cam_id
        self.width = width
        self.height = height
    def get_img(self):
        IpLastSegment = "123"
```

```
        cam = self.cam_id
        udpstrPrevData = "udpsrc address=192.168.123."+ IpLastSegment + " port="
        udpPORT = [9201,9202,9203,9204,9205]
        udpstrBehindData = " ! application/x-rtp,media=video,encoding-name=H264 !
rtph264depay ! h264parse ! avdec_h264 ! videoconvert ! appsink"
        udpSendIntegratedPipe_0 = udpstrPrevData +  str(udpPORT[cam-1]) +
udpstrBehindData
        print(udpSendIntegratedPipe_0)


        self.cap = cv2.VideoCapture(udpSendIntegratedPipe_0)


    def demo(self):
        self.get_img()
        while(True):
            self.ret, self.frame = self.cap.read()
            self.frame = cv2.resize(self.frame, (self.width, self.height))
            if self.cam_id == 1:
                self.frame = cv2.flip(self.frame, -1)
            if self.frame is not None:
                cv2.imshow("video0", self.frame)
            if cv2.waitKey(2) & 0xFF == ord('q'):
                break
        self.cap.release()
        cv2.destroyAllWindows()
```

Note: Using pip3 to install opencv-python and opencv-contrib-python will be missing dependencies, be sure to source install opencv