3D Camera Face Detection and Distance Measurement

I. Experiment Objectives

1. To master face detection and distance measurement using a 3D camera, by real-time detecting faces in video and calculating the distance between the face and the camera based on depth information.

2. To master camera data fusion and display, processing and stitching color and depth images, enabling multi-dimensional information visualization.

II. Experiment Content

The 3D camera measures the distance to a face based on depth images captured by the ORBBEC DaBai depth camera.

Experiment workflow:

1. Use the ORBBEC DaBai depth camera to capture RGB and depth image data.
2. Use OpenCV's Haar feature detector to detect faces in the color image.
3. Combine depth data from the depth image to calculate the distance from the face to the camera.
4. Stitch the RGB and depth images together for display, achieving an intuitive effect.
5. Control the stepper motor to adjust the camera's direction.

III. Experiment Environment

Operating System	Linux ARM64
Experimental Equipment	AI Experiment Box
Experimental Accessories	Depth Camera

IV. Experiment Principle

1.Hardware Principle

The ORBBEC DaBai hardware includes two depth cameras, one RGB camera, and a structured light infrared projector. The depth camera uses line-scanning, and the infrared structured light measures distance.

2.Algorithm Principle

This experiment captures the face image using the RGB camera and detects the face, generating a detection box. The center point of the detection box is then mapped to the depth image, obtaining the depth value corresponding to the face center, which represents the distance from the camera.

3.Related Functions

Control gimbal rotation function: blinx_bus_stepping_motor() function. This function is a packaged function that controls the gimbal's rotation by sending a hex array via serial communication. The 5th and 6th positions in the array correspond to the angle values for motor rotation. Refer to (6. Embedded Systems and Applications / 11 Gyroscope-Based Gimbal Control) for specific gimbal motor control.

Experiment Flow

(1) Face Detection: Use OpenCV's Haar feature cascade classifier to detect faces in the RGB image. Haar features analyze the image's grayscale distribution patterns to identify faces.

(2) Depth Measurement: The 3D camera measures depth information by calculating the time taken for infrared light to reflect off an object's surface. Combined with the face detection result, the corresponding depth data for the location is mapped to obtain the face's distance from the camera.

(3) Image Processing: Process the depth image and color image, including normalization and pseudocolor mapping, to make the depth data visually intuitive.

(4) Stepper Motor Control: Control the position of the stepper motor by sending control commands via serial communication, adjusting the camera's direction.

V. Experiment Steps

1.Run Jupyter Lab

(1) Open the "Experiment" folder on the desktop, right-click on the empty space, then select “Open in Terminal.” In the terminal, type `jupyter lab`.

(2) In Jupyter Lab, choose Python3 under Notebook, and enter the program editor.

(3) Right-click on the “Untitled.ipynb” notebook, select "Rename," and name the notebook according to the experiment title.

(4) In the new cell, input `%load 3D Camera Face Detection and Distance Measurement.py` and click the Run button to load the program.

2.Import Libraries and Functions

Import various libraries and modules required in the code for camera data processing, image processing, numerical computation, serial communication, and time control.

Reference Code :

Create a new program block named "Camera Face Detection and Distance Measurement.ipynb"

```
from pyorbbecsdk import *    # Import Orbbec SDK
import cv2    # Import OpenCV for image processing
import numpy as np    # Import NumPy for numerical processing
from utils import frame_to_bgr_image # Import custom image conversion tool
import sys    # Import sys module for system operations
import serial    # Import serial module for serial communication
import time    # Import time module for time delay operations
```

3.Initialize Serial Communication

Initialize the serial communication for the gimbal device, set the baud rate to 115200, and timeout to 5 seconds.

Reference Code :

Continue writing in the "Camera Face Detection and Distance Measurement.ipynb" program block.

```
# Initialize serial communication, set baud rate to 115200, timeout to 5 seconds
data_ser = serial.Serial("/dev/user_robot", 115200, timeout=5)
```

4.Gimbal Control Function

Define a function to adjust the stepper motor position by sending control commands via serial.

Reference Code :

Continue writing in the "Camera Face Detection and Distance Measurement.ipynb" program block

```
def blinx_bus_stepping_motor(value):    # Define stepper motor control function
    ddata = [0xFF, 0xFE, 0x02, 0x01, 0x00, 0x01, 0x46, 0x00, 0x0D, 0x0A]   # 
Initialize control data array
    ddata[5] = value    # Set motor position value
    data_ser.write(ddata)    # Send data via serial
    time.sleep(0.1)    # Wait for 0.1 seconds to ensure the data is sent
```

5.Camera Initialization Function

Initialize camera settings and data stream pipeline, enable both color and depth streams, and return initialized pipeline and config objects.

Reference Code :

Continue writing in the "Camera Face Detection and Distance Measurement.ipynb" program block.

```
ef blinx_initialize_camera_pipeline():    # Define camera initialization function
    pipeline = Pipeline()    # Create a Pipeline object for camera data stream processing
    config = Config()    # Create a Config object for camera parameter configuration
    try:
        color_profile_list = 
pipeline.get_stream_profile_list(OBSensorType.COLOR_SENSOR)    # Get color image 
stream configuration
        color_profile = color_profile_list.get_default_video_stream_profile()    # Get 
default video stream profile
        config.enable_stream(color_profile)    # Enable color stream
        depth_profile_list = 
pipeline.get_stream_profile_list(OBSensorType.DEPTH_SENSOR)    # Get depth image 
stream configuration
        depth_profile = depth_profile_list.get_default_video_stream_profile()    # Get 
default depth video stream profile
        config.enable_stream(depth_profile)    # Enable depth stream
        print("color profile : {}x{}@{}_{}".format(    # Output color stream configuration
            color_profile.get_width(), color_profile.get_height(),
            color_profile.get_fps(), color_profile.get_format()))
        print("depth profile : {}x{}@{}_{}".format(    # Output depth stream configuration
            depth_profile.get_width(), depth_profile.get_height(),
            depth_profile.get_fps(), depth_profile.get_format()))
    except Exception as e:    # Catch initialization exceptions
```

```
        print("Camera initialization failed:", e)    # Print error information
        return None, None    # Return None if initialization fails
    return pipeline, config    # Return initialized pipeline and config objects
```

6.Start Camera Function

Start the camera data stream and load the face detector, returning the face detector object.

Reference Code :

Continue writing in the "Camera Face Detection and Distance Measurement.ipynb" program block

```
def blinx_start_camera(pipeline, config):    # Define camera start function
    try:
        pipeline.start(config)    # Start pipeline for data stream processing
        face_detector = cv2.CascadeClassifier("haarcascade_frontalface_default.xml")
# Load face detector
    except Exception as e:    # Catch startup exceptions
        print("Failed to start camera:", e)    # Print error information
        return None    # Return None if startup fails
    return face_detector    # Return the face detector object
```

7.Depth Image Frame Processing Function

Process depth image frames by converting depth data to floating-point values, normalizing, and applying pseudocolor mapping.

Reference Code :

Continue writing in the "Camera Face Detection and Distance Measurement.ipynb" program block.

```
def blinx_process_depth_frame(depth_frame):    # Define depth image frame processing
function
    width = depth_frame.get_width()    # Get depth image width
    height = depth_frame.get_height()    # Get depth image height
    scale = depth_frame.get_depth_scale()    # Get depth scale
    depth_data = np.frombuffer(depth_frame.get_data(), dtype=np.uint16)    #    Convert
depth data to NumPy array
    depth_data = depth_data.reshape((height, width))    # Reshape array to image size
    depth_data = depth_data.astype(np.float32) * scale    # Convert depth data to float and
apply scale
    depth_image = cv2.normalize(depth_data, None, 0, 255, cv2.NORM_MINMAX,
dtype=cv2.CV_8U)    # Normalize depth data
    depth_image = cv2.applyColorMap(depth_image, cv2.COLORMAP_JET)    #
Normalize depth data
    return depth_image, depth_data    # Apply pseudocolor
```

8.Define a Function to Display Image Frames

Display the concatenated effect of color images and depth images, detect faces, and draw face bounding boxes and distance information on the image.

Reference Code :

Continue writing the program block "Camera Face Detection and Distance Measurement.ipynb".

```
def blinx_display_frames(color_image, depth_image, depth_data, face_detector):    # Define a function to display image frames
    gray = cv2.cvtColor(color_image, cv2.COLOR_BGR2GRAY)    # Convert the color image to grayscale
    faces = face_detector.detectMultiScale(gray, 1.2, 3)    # Detect faces, adjust scale and minNeighbors
    for x, y, w, h in faces:    # Iterate through detected face regions
        cv2.rectangle(color_image, (x, y), (x + w, y + h), (0, 0, 255), 2)    # Draw a red rectangle around the face
        object_x = round(x + w / 2)    # Calculate the x-coordinate of the face center
        object_y = round(y + h / 2)    # Calculate the y-coordinate of the face center
        if object_y <= 400:    # If the y-coordinate is within the depth image range
            center_distance = depth_data[object_y, object_x]    # Get the depth value of the face center
            cv2.putText(color_image, "dst:" + str(round(center_distance)) + "mm",    # Display distance information on the image
                        (x, y - 5), cv2.FONT_HERSHEY_SIMPLEX, 0.5, [0, 255, 255])
    h0, w0 = depth_image.shape[0], depth_image.shape[1]    # Get the dimensions of the depth image
    h1, w1 = color_image.shape[0], color_image.shape[1]    # Get the dimensions of the color image
    h = max(h0, h1)    # Calculate the height of the concatenated image
    w = max(w0, w1)    # Calculate the width of the concatenated image
    org_image = np.ones((h, w, 3), dtype=np.uint8) * 255    # Create a blank image for the depth image
    trans_image = np.ones((h, w, 3), dtype=np.uint8) * 255    # Create a blank image for the color image
    org_image[:h0, :w0, :] = depth_image[:, :, :]    # Fill the blank image with the depth image
    trans_image[:h1, :w1, :] = color_image[:, :, :]    # Fill the blank image with the color image
    all_image = np.hstack((org_image[:, :w0, :], trans_image[:, :w1, :]))    # Concatenate the
```

```
depth and color images horizontally
    cv2.imshow("compare image", all_image)   # Display the concatenated image
```

9. Get and Process Frames from the Camera

Get and process the frames captured from the camera, process both color and depth image frames, and return the processed results and status flag.

Reference Code :

Reference Code:Continue writing the program block "Camera Face Detection and Distance Measurement.ipynb".

```
def blinx_get_processed_frames(pipeline):# Get and process frames from the camera
    try:
        # Wait for frames with a timeout of 100ms
        frames: FrameSet = pipeline.wait_for_frames(100)
        if frames is None:
            return None, None, None, False   # No frames obtained
        # Get the color image frame
        color_frame = frames.get_color_frame()
        if color_frame is None:
            return None, None, None, False   # No RGB frame obtained
        # Convert the frame to a BGR image
        color_image = frame_to_bgr_image(color_frame)
        if color_image is None:
            print("Failed to convert frame to image")
            return None, None, None, False   #   Frame conversion failed
        # Get the depth image frame
        depth_frame = frames.get_depth_frame()
        if depth_frame is None:
            return None, None, None, False   # No depth frame obtained
        # Process the depth image frame
        depth_image, depth_data = blinx_process_depth_frame(depth_frame)
        if depth_image is None:
            return None, None, None, False   # Depth image processing failed
        return color_image, depth_image, depth_data, True   # Return the processed data
and success flag
    except Exception as e:
        print(f"An error occurred while processing frames: {e}")
        return None, None, None, False   # Catch exceptions and return error flag
```

10. Define the Main Program

Execute the main program flow, control the stepper motor, initialize the camera, process video frames, display images, respond to exit conditions, and clean up resources.

Reference Code :

Continue writing the program block "Camera Face Detection and Distance Measurement.ipynb".

```
def main(argv):
    blinx_bus_stepping_motor(0x5A)    # Call the stepper motor control function to set the
motor position
    pipeline, config = blinx_initialize_camera_pipeline()    # Initialize the camera
    if pipeline is None or config is None:    # If camera initialization fails, exit the program
        return
    face_detector = blinx_start_camera(pipeline, config)    # Start the camera and load the
face detector
    if face_detector is None:    # If camera startup fails, exit the program
        return
        return
    while True:    # Infinite loop for real-time video frame processing
        color_image, depth_image, depth_data, success =
blinx_get_processed_frames(pipeline)
        if not success:
            continue    # Skip this loop if frame data acquisition or processing fails
        blinx_display_frames(color_image, depth_image, depth_data, face_detector)    #
Detect faces and display the image
        key = cv2.waitKey(1)    # Wait for key press event with a delay of 1ms
        if key == ord('q') or key == 27:    # If 'q' or ESC key is pressed, exit the loop
            break
    pipeline.stop()    # Stop the pipeline and end the data stream processing
    cv2.destroyAllWindows()    # Destroy all OpenCV windows
```

11. Run the Main Program

Check if the program is running as the main module, and if so, start the main function to execute the main flow.

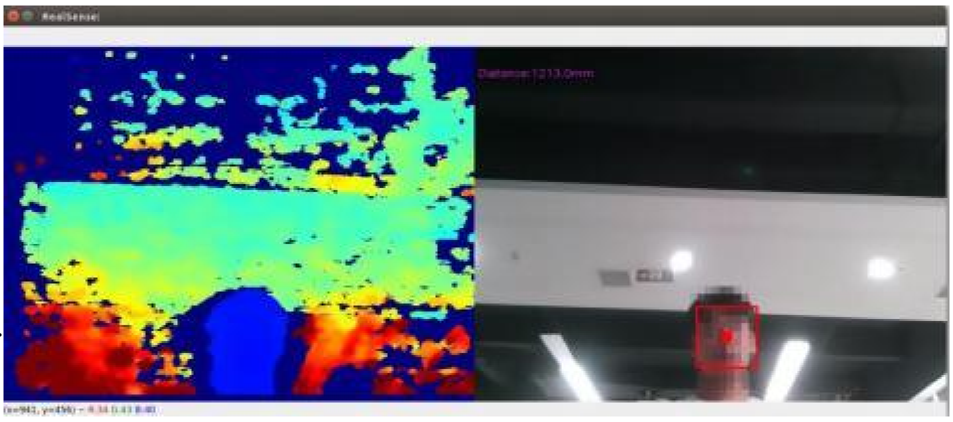
Reference Code :

Continue writing the program block "Camera Face Detection and Distance Measurement.ipynb".

```
if __name__ == "__main__":
    main(sys.argv[1:])    # Execute the main function
```

12. Run the Main Program Code Effect

The following shows the effect of running the main program


RealSense
Distance:1213.0mm