

Công ty cổ phần kỹ thuật Temas

REPORT NỘI DUNG TASK 1
DETECT PALLET USING ARUCO MARKER

Người thực hiện: Phạm Thị Mỹ Lệ

Thời gian: 6 tháng

HÀ NỘI – 2024

MỤC LỤC

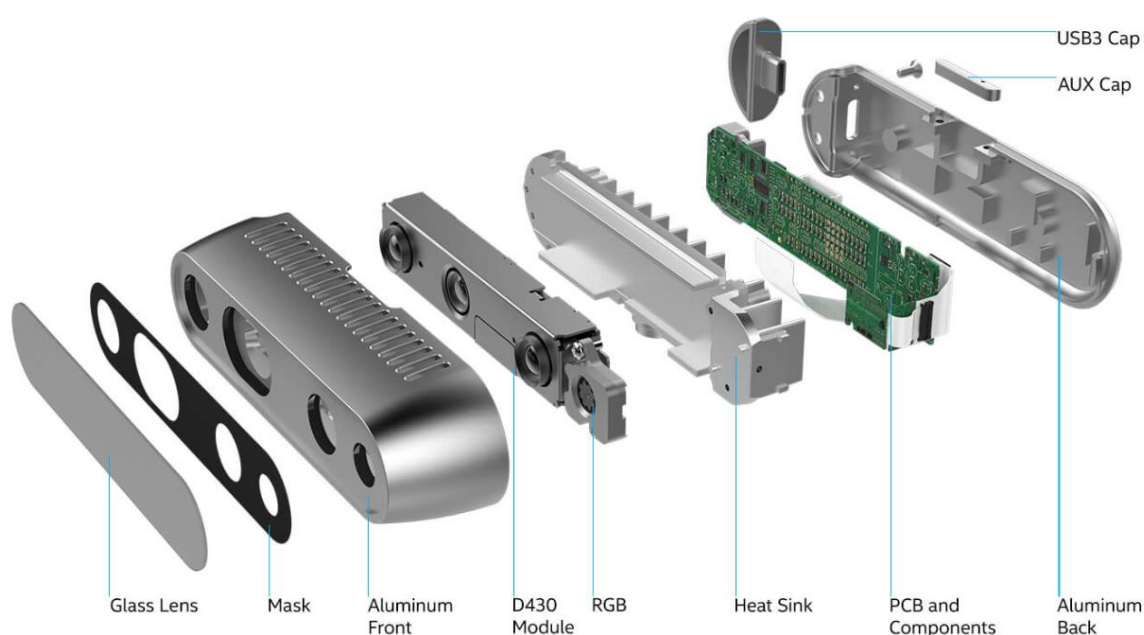
PHẦN I. SỬ DỤNG CAMERA NHẬN DIỆN GIÁ HÀNG	4
I. Sơ lược về Camera D435/D435f	4
II. Thuật toán sử dụng	5
1. Bài toán đặt ra.....	5
2. RANSAC kết hợp Point Cloud 3D	7
2.1 Point Cloud	7
2.2 RANSAC	9
2.3 Thực thi bài toán	10
3. ArUco Marker.....	14
3.1 ArUco Marker Detection.....	14
3.2 Algorithm flowchart	15
4. Đánh giá sai số của thuật toán	18
4.1 Đánh giá sai số	18
4.2 Hiệu chỉnh sai số.....	19
4.3 Thay đổi thiết bị D435f.....	21
4.4 Đánh giá sai số của D435f.....	24
5. Kết luận và chiến lược thực thi bài toán	25
PHẦN II. ORANGE PI 4 VÀ GIAO THỨC TRUYỀN THÔNG TCP/IP ...	26
I. Sơ đồ hệ thống.....	26
II. Orange Pi 4.....	27
III. Giao thức TCP/IP	29
1. Nguyên lý hoạt động	29
2. Triển khai hệ thống	31
PHẦN III. THIẾT KẾ GIAO DIỆN THÔNG QUA QT CREATOR.....	34
I. Giới thiệu tổng quan về QT Creator	34

II. Cấu trúc dự án	35
III. Tích hợp hệ thống.....	38
PHẦN IV. KẾT LUẬN.....	39

PHẦN I. SỬ DỤNG CAMERA NHẬN DIỆN GIÁ HÀNG

I. Sơ lược về Camera D435/D435f

Camera Intel RealSense D435 là một loại camera độ sâu của Intel thuộc dòng D400, được thiết kế để ghi lại thông tin 3D chất lượng cao. Camera này sử dụng công nghệ stereo kết hợp với cảm biến RGB và máy chiếu tia laser hồng ngoại để tạo ra bản đồ độ sâu chính xác. Nhờ khả năng hoạt động tốt trong nhiều điều kiện ánh sáng khác nhau, D435 có thể ghi hình với tốc độ lên đến 90 fps, giúp thu thập dữ liệu mượt mà và chính xác.



Intel RealSense SDK 2.0 ngoài ra còn cung cấp tính năng tự hiệu chuẩn trên chip cho thuộc dòng camera stereo D400. Tính năng này cho phép người dùng hiệu chuẩn camera một cách nhanh chóng, chỉ trong vòng chưa đầy 15 giây, mà không cần đến các mục tiêu chuyên dụng phức tạp. Quá trình hiệu chuẩn tự động này giúp giảm bớt các bước phức tạp và tăng cường độ chính xác của camera, đảm bảo rằng các thiết bị luôn được tối ưu hóa để cung cấp dữ liệu độ sâu chất lượng cao trong mọi điều kiện sử dụng. Do đó người dùng hoàn toàn không cần phải thực hiện việc các bước calib cơ bản như các camera trước đây.

Dưới đây là bảng các thông số kỹ thuật của D435:

Use Environment	Indoor/Outdoor
Depth Technology	Active IR stereo
Main Intel® RealSense™ component	- Intel® RealSense™ Vision Processor D4 - Intel® RealSense™ module D430
Depth Field of View (FOV) (Horizontal x Vertical x Diagonal)	86° x 57° (±3)
Depth Stream Output Resolution	Up to 1280 x 720
Depth Stream Output Frame Rate	Up to 90 fps
Minimum Depth Distance (Min-Z)	0.1 m
Sensor Shutter Type	Global Shutter
Maximum Range	Approx. 10 meters; Varies depending on calibration, scene, and lighting condition
RGB Sensor Resolution and Frame Rate	1920 x 1080 at 30 fps
RGB Sensor FOV (Horizontal x Vertical x Diagonal)	69.4° x 42.5° x 77° (+/- 3°)
Camera Dimension (Length x Depth x Height)	90 mm x 25 mm x 25 mm
Connectors	USB-C* 3.1 Gen 1
Mounting Mechanism	- One 1/4-20 UNC thread mounting point - Two M3 thread mounting points

Chính vì các giá trị thông số và cấu hình của Camera Intel RealSense D435, đây là lựa chọn lý tưởng cho dự án nhận diện giá hàng ứng dụng trong Robot di động. Nhờ và việc thu thập dữ liệu độ sâu chính xác trong không gian 3D và trường nhìn rộng của D435 cho phép bao quát toàn bộ giá hàng một cách dễ dàng, trong khi cảm biến màn trập toàn cục đảm bảo hiệu suất tốt trong nhiều điều kiện ánh sáng, điều này rất quan trọng khi làm việc trong các môi trường kho chứa hàng. Khả năng tự hiệu chuẩn nhanh chóng cũng giúp duy trì độ chính xác cao mà không cần can thiệp thủ công, giúp tối ưu hóa hiệu suất nhận diện và điều hướng của robot.

II. Thuật toán sử dụng

1. Bài toán đặt ra

Mục tiêu của dự án này là xây dựng một hệ thống giúp robot nhận diện được vị trí của giá hàng, tính toán ra được điểm chính giữa của giá và điều khiển di chuyển cho Robot nâng đỡ giá hàng.



Hình minh họa giá hàng trong thực tế

Để đạt được mục tiêu này, em sẽ giải quyết bài toán nhận diện hai marker màu xanh lá được gắn trên giá hàng. Thay vì chỉ dựa vào ảnh RGB cơ bản, để gây nhiễu do các màu tương tự, em sẽ tận dụng khả năng của camera D435 để nhận diện marker dựa trên dữ liệu điểm 3D (point cloud). Bằng cách lọc các điểm không thỏa mãn tiêu chí độ sâu, hệ thống sẽ xác định chính xác vị trí của các marker. Chi tiết các bước triển khai sẽ được trình bày trong các phần 2.



Hình minh họa marker được đánh dấu

2. RANSAC kết hợp Point Cloud 3D

2.1 Point Cloud

Point cloud là tập hợp các điểm dữ liệu rời rạc trong không gian. Các điểm điểm này có thể biểu diễn hình dạng của các vật thể 3D. Mỗi điểm biểu thị tọa độ Descartes (x, y, z) và có thể chứa các dữ liệu khác như: màu RGB, pháp tuyến, dấu thời gian... Point cloud là một phương pháp phổ biến để mô tả hình dạng của bề mặt các đối tượng trong môi trường ba chiều, nhờ vào khả năng cung cấp thông tin chi tiết và chính xác về vị trí và không gian của các điểm.

Point cloud có thể được thu thập bằng các thiết bị quét 3D, camera độ sâu như Intel RealSense D435, hoặc cảm biến LiDAR. Đối với camera D435, quá trình thu thập point cloud được thực hiện bằng cách sử dụng cảm biến độ sâu để đo khoảng cách từ camera đến các điểm trên bề mặt của đối tượng trong không gian. Dữ liệu thu được bao gồm hàng triệu điểm riêng lẻ, tạo thành một bản đồ chi tiết về không

gian xung quanh. Với công nghệ của Intel RealSense, người dùng có thể lọc và phân tích dữ liệu để nhận diện các đối tượng và tính toán thông tin chi tiết về chiều sâu, giúp robot hoặc các hệ thống tự động hóa hiểu và tương tác chính xác với môi trường.

Trong quá trình lập trình, thư viện Realsense2 sẽ giúp thu thập dữ liệu point cloud từ D435 và lưu trữ trong file định dạng .ply (Polygon File Format) hoặc file .PCD (Point Cloud Data).

```
# Lấy thông tin camera
intrinsics = depth_frame.profile.as_video_stream_profile().intrinsics

# Chuyển đổi depth frame thành point cloud
pc = rs.pointcloud()
pc.map_to(depth_frame)
point_cloud = pc.calculate(depth_frame)

# Lưu trữ point cloud dưới định dạng .ply
point_cloud.export_to_ply("point_cloud.ply", depth_frame)
```

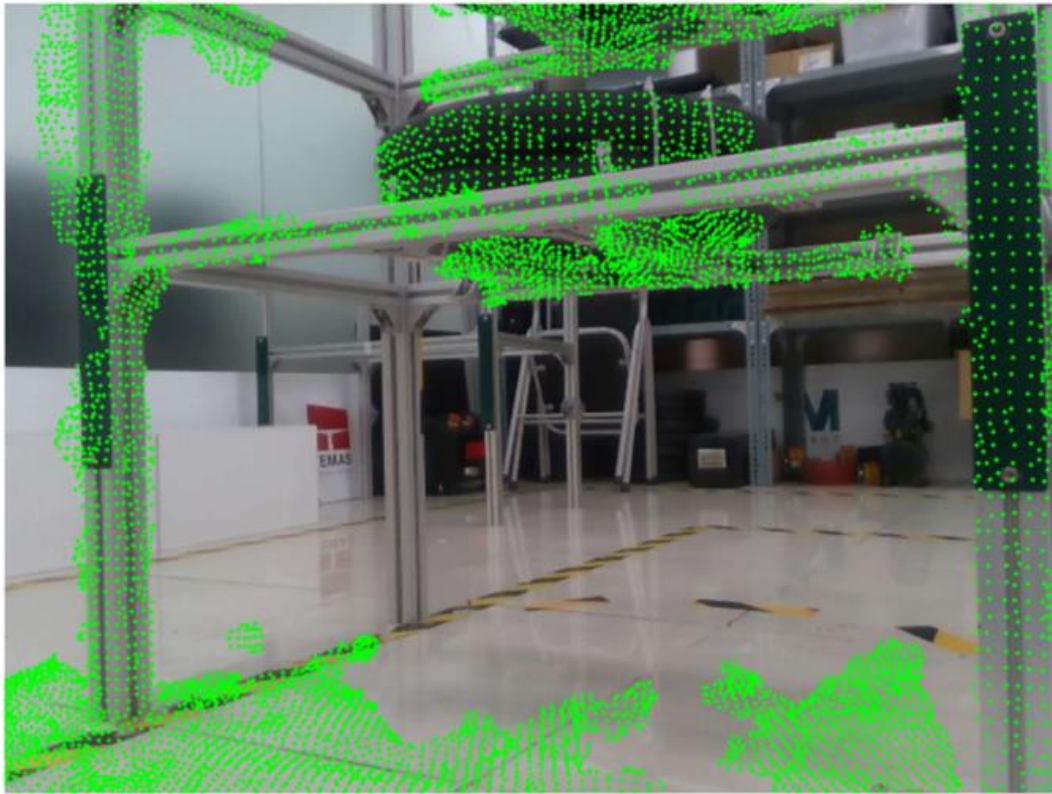
Code minh họa thu thập point cloud

Trong một Point Cloud, mỗi điểm dữ liệu (point) thường chứa các giá trị sau:

- **Tọa độ không gian (x, y, z):** Đây là giá trị chính của mỗi điểm, biểu thị vị trí của điểm đó trong không gian 3D.
- **Màu sắc (RGB):** Một số point cloud bao gồm cả giá trị màu sắc, biểu thị màu sắc của điểm đó trong không gian.
- **Pháp tuyến (Normals):** Một số point cloud có thể chứa thông tin về pháp tuyến tại mỗi điểm, dùng để mô tả hướng của bề mặt tại điểm đó.
- **Cường độ (Intensity):** Giá trị cường độ ánh sáng phản xạ, thường được sử dụng trong các cảm biến như LiDAR.

Dưới đây là kết quả thu được của Point Cloud khi chỉ giữ lại các điểm có khoảng cách < 2m mà D435 thu thập được từ giá để hàng. Có thể thấy được, các giá trị này chứa

rất nhiều giá trị nhiễu, do đó các thuật toán, lọc là điều rất cần trong trường hợp này.



Hình minh họa point cloud thu trong phạm vi 2m

2.2 RANSAC

RANSAC, viết tắt của “RANdom SAmple Consensus”, là một phương pháp lặp được sử dụng để ước tính các tham số của một mô hình toán học từ một tập dữ liệu chứa cả giá trị nội tại (inlier) và giá trị ngoại lai (outlier). Mục đích chính của RANSAC là xác định và tách biệt các giá trị nội tại để tìm ra mô hình phù hợp nhất với dữ liệu.

Quy trình hoạt động của RANSAC

- 1. Chọn ngẫu nhiên:** Chọn ngẫu nhiên một tập nhỏ các điểm dữ liệu.
- 2. Ước lượng mô hình:** Tính toán mô hình từ các điểm đã chọn.
- 3. Xác định inlier:** Kiểm tra các điểm khác trong tập dữ liệu, xác định bao nhiêu điểm phù hợp với mô hình đã tính toán trong một ngưỡng sai số cho trước.
- 4. Lặp lại:** Lặp lại quá trình trên một số lần cố định.
- 5. Chọn mô hình tốt nhất:** Giữ lại mô hình với số lượng điểm nội tại (inlier)

nhiều nhất hoặc mô hình có lỗi thấp nhất.

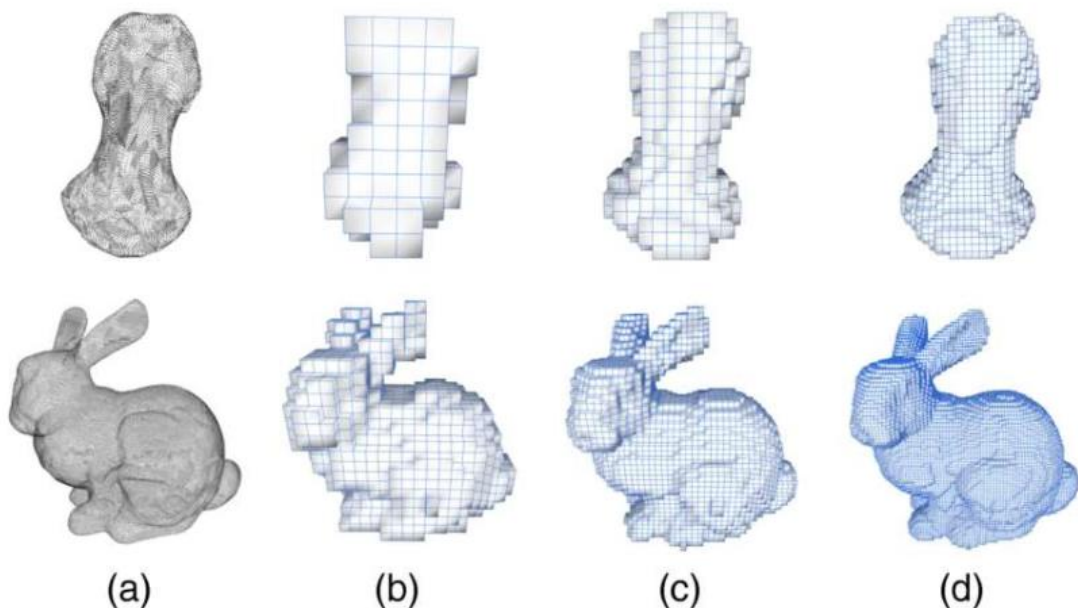
Một trong những lợi thế của RANSAC là khả năng ước tính mạnh mẽ các tham số của mô hình, tức là có thể ước tính được các tham số với độ chính xác cao ngay cả khi có một số lượng lớn các giá trị outlier trong tập dữ liệu. Nhưng bù lại, RANSAC không có giới hạn trên về thời gian cần thiết để tính toán các tham số này, khi số lần lặp tính toán được giới hạn, giải pháp thu được có thể không tối ưu và thậm chí không đưa ra được kết quả tốt nhất.

2.3 Thực thi bài toán

Quá trình tiền xử lý point cloud là bước quan trọng để chuẩn bị cho bước nhận diện và phân tích tiếp theo. Các bước tiền xử lý ở đây bao gồm: giảm dữ liệu, chuyển đổi không gian màu, lọc dữ liệu theo ngưỡng màu, ...

- **Giảm dữ liệu với Voxel Grid**

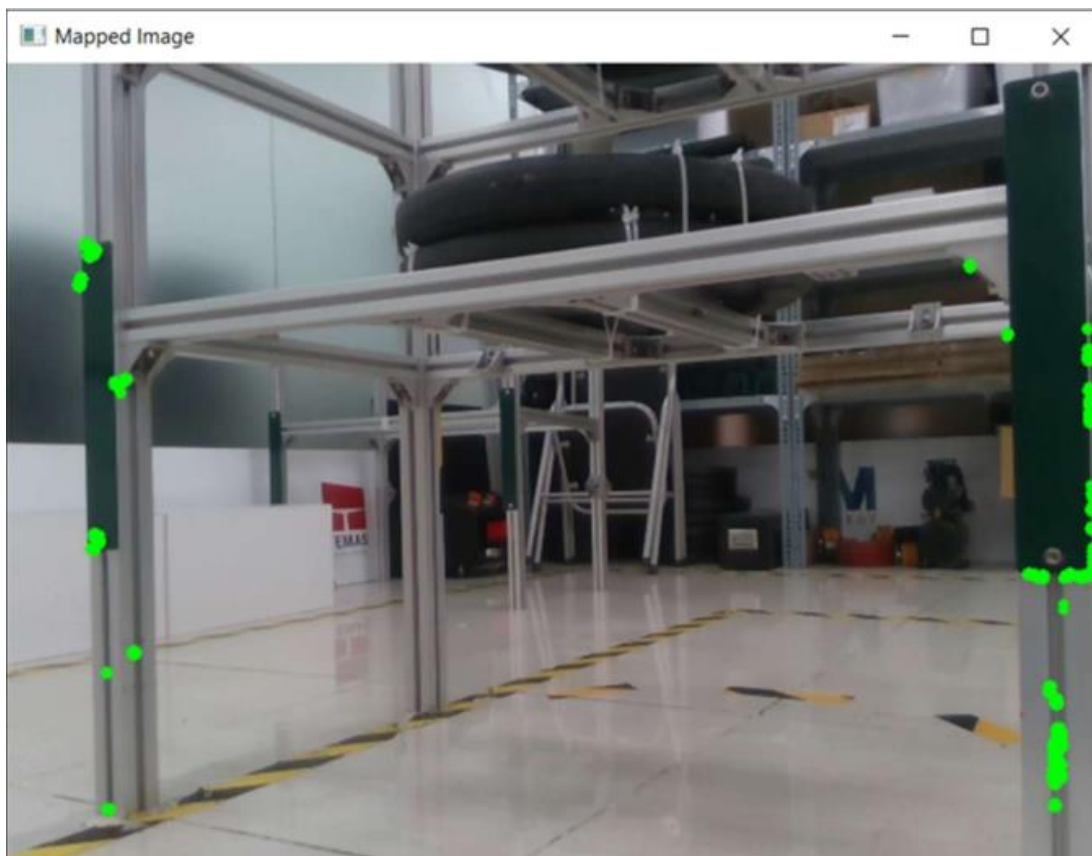
Đầu tiên, phương pháp Voxel Grid được sử dụng để giảm số lượng điểm trong point cloud. Phương pháp thực hiện bằng cách chia không gian 3D thành các ô nhỏ (gọi là voxel) và chỉ giữ lại một điểm đại diện cho mỗi ô, giúp giảm đáng kể số lượng điểm mà vẫn giữ được cấu trúc tổng quát của dữ liệu. Điều này giúp tăng tốc độ xử lý trong các bước tiếp theo mà không làm mất nhiều thông tin quan trọng.



Voxel

- **Chuyển đổi không gian màu từ RGB sang HS**

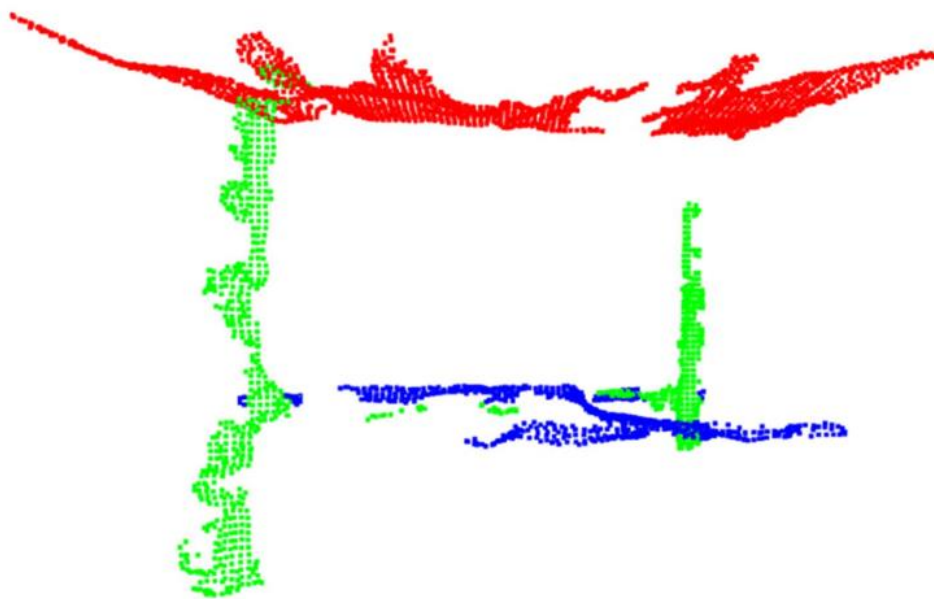
Ngoài dữ liệu vị trí x, y, z , point cloud còn chứa thông tin về màu sắc của từng điểm. Ban đầu, em sử dụng không gian màu RGB để lọc các điểm có màu xanh. Tuy nhiên, do màu sắc trong không gian RGB chịu ảnh hưởng nhiều từ điều kiện ánh sáng, nên em quyết định chuyển sang không gian màu HSV. HSV phân tích thông tin sắc thái và độ bão hòa của mẫu khỏi thông tin độ sáng, nên sẽ hữu ích hơn trong không gian mà cường độ ánh sáng thay đổi ngẫu nhiên.



Hình minh họa lọc theo ngưỡng RGB nhưng còn khá nhiều nhiễu

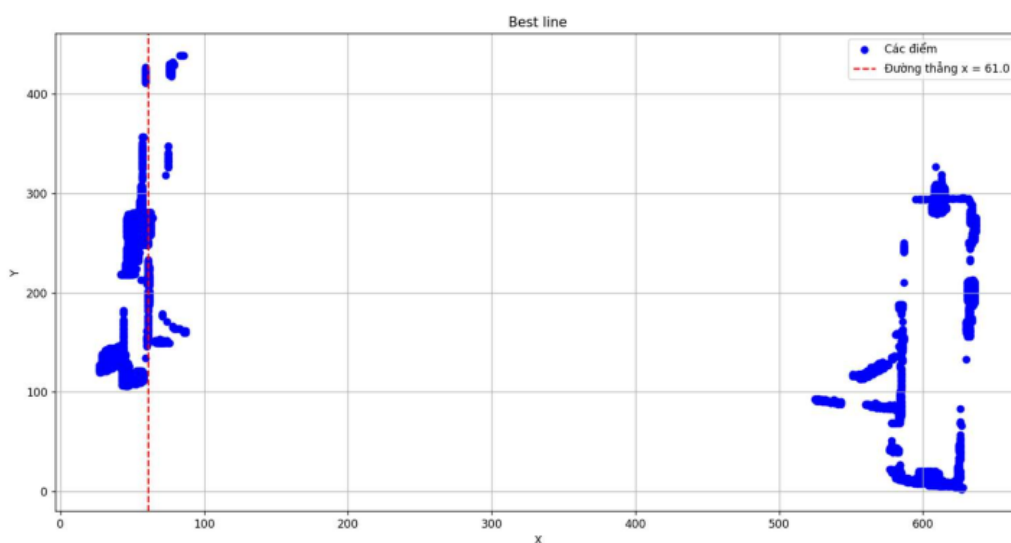
- **Lọc mặt phẳng thẳng đứng chứa cạnh biên bằng RANSAC**

Sau khi lọc giữ lại các điểm có màu xanh, thuật toán RANSAC được tiến hành để nhận diện các mặt phẳng. Ở đây, mặt phẳng thẳng đứng là mặt phẳng chứa cạnh biên của giá để hàng, nên RANSAC sẽ chỉ giữ lại mặt phẳng đó.



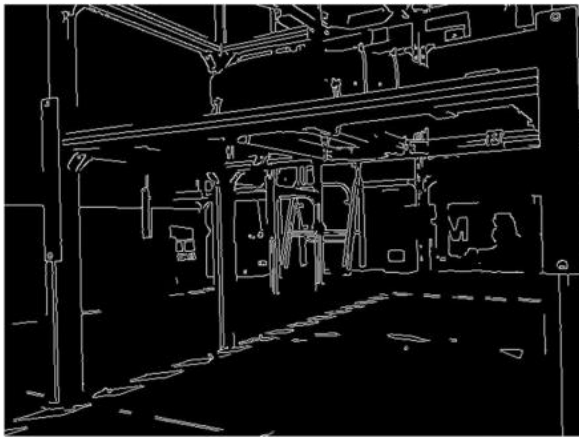
Lọc mặt phẳng

Hình trên minh họa 3 mặt phẳng tách biệt của giá để hàng thông qua RANSAC. Bao gồm có mặt phẳng đất (blue), mặt phẳng marker thẳng đứng (green), mặt phẳng trên (red). Sau khi giữ lại được mặt phẳng rồi, điều quan trọng là làm thế nào để có thể nhận diện ra được các cạnh biên của marker. Nếu từ tập hợp point cloud cho trước, ta tìm phương trình đường thẳng thẳng đứng song song với Oy đi qua tập các điểm point cloud cho trước thì bài toán sẽ chỉ đảm bảo được tính đúng đắn khi mà camera được đặt thẳng thẳng đứng. Còn khi Camera đặt lệch góc thì kết quả tính toán sẽ không còn đúng.

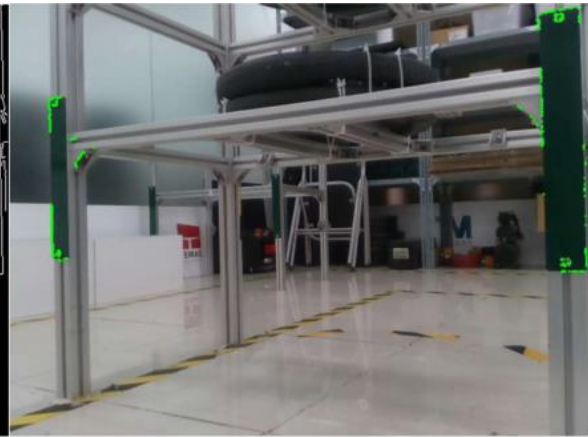


Point Cloud marker trong không gian 3D

Để có thể phù hợp với mọi trạng thái của camera, em đề xuất phương pháp kết hợp 3D point cloud và edge 2D. Tức là từ point cloud trên mặt phẳng được giữ lại, em sẽ kết hợp với ảnh 2D về các vector cạnh được lọc thông qua Sobel, Canny ... để tìm ra được các cạnh tạo lên từ các pointcloud.



Hình 10a: Ảnh 2D sau khi sử dụng bộ lọc Canny để tìm cạnh



Hình 10b: Point Cloud còn lại sau khi kết hợp giữa RANSAC và Edge image

2.4 Kết quả



Tọa độ 4 điểm trên 4 góc của marker

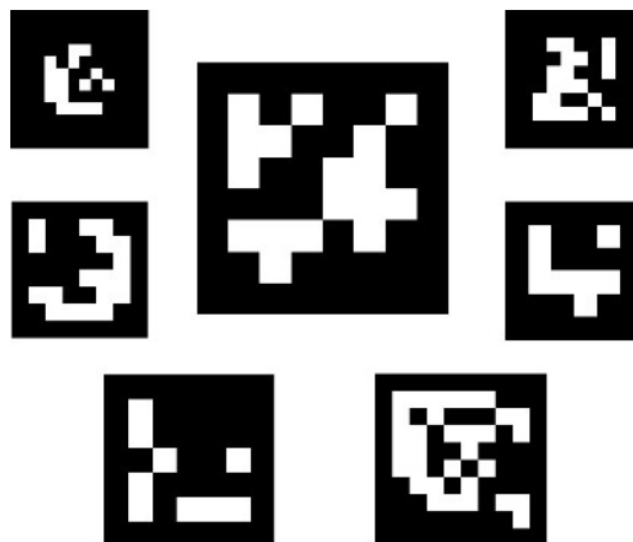
Thông qua đó, ta có thể tìm ra được 4 điểm ở 4 góc của marker. Từ 4 điểm này ta có thể tìm ra được điểm chính giữa của 2 marker, đồng thời chuyển đổi tọa độ camera sang tọa độ thực và đánh giá kết quả, sai số.

Tuy nhiên, do D435 còn khá nhiều, và việc thu thập file .ply trong 1 khoảng thời gian ngắn, nên các point cloud có sự không chính xác thậm chí là lệch và nhiễu khá nhiều. Nếu tiếp tục theo phương pháp RANSAC, thì tỉ lệ lấy ra được chính xác tọa độ của 4 góc là rất thấp. Do đó cần tìm hiểu thêm các phương pháp khác phù hợp hơn.

3. ArUco Marker

3.1 ArUco Marker Detection

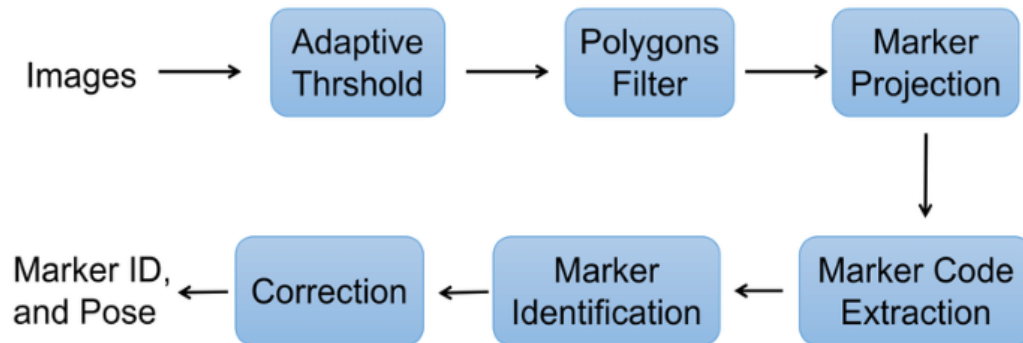
ArUco marker là một điểm đánh dấu được tạo bởi một hình vuông có đường viền đen và ma trận nhị phân bên trong xác định mã định danh của nó. Để phân biệt các ArUco marker khác nhau, hệ thống sử dụng **mã nhị phân** độc đáo của mỗi marker. Mã này được xác định dựa trên một dictionary chứa tất cả các mã marker có thể có. Khi một marker được phát hiện, mã nhị phân của nó được trích xuất và so sánh với các mã trong dictionary để xác định ID của marker. Mỗi ID là duy nhất trong dictionary, do đó, mỗi marker có thể được phân biệt rõ ràng, ngay cả khi chúng có kích thước hoặc hình dạng tương tự.



ArUco Marker

Sơ đồ bên dưới là cách để có thể nhận diện được marker nhị phân. Đầu tiên, từ ảnh đầu vào, chương trình sẽ phân tích những hình vuông khả nghi trong ảnh, sau đó, trích xuất các đường viền và chỉ giữ lại hình vuông. Sau đó chuẩn hóa mã nhị phân của marker

và đối chiếu với Dictionary để xác định ID của nó. Kết quả sẽ trả về tọa độ của các marker.



Sơ đồ nhận diện ArUco Marker

Có một số hàm có sẵn để có thể giúp phát hiện marker nhanh hơn:

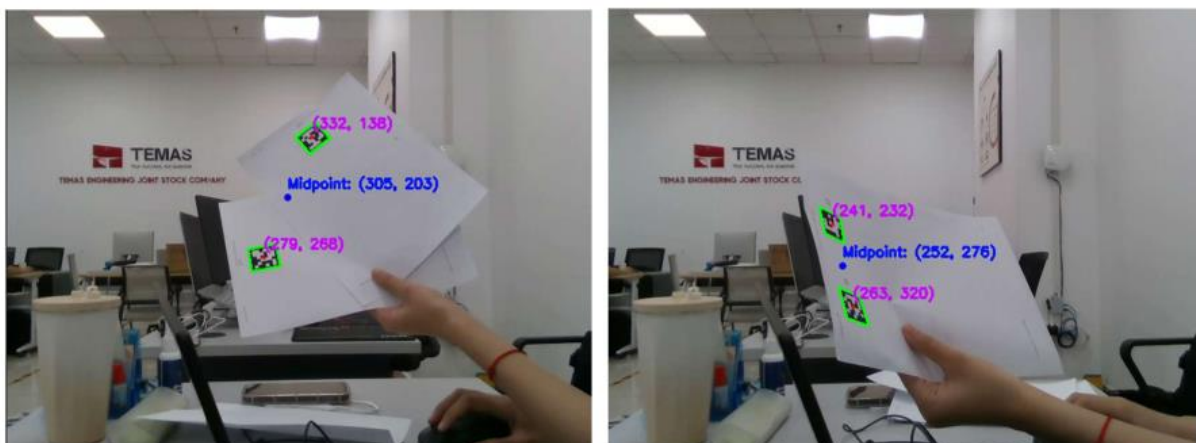
cv::aruco::detectMarkers(): Hàm này dùng để phát hiện các markers trong ảnh. Nó yêu cầu ảnh đầu vào, dictionary của marker để đối chiếu, và trả về danh sách các góc của markers được phát hiện và ID tương ứng.

cv::aruco::getPredefinedDictionary(): Hàm này trả về một dictionary được định nghĩa trước, chứa các ID và mẫu nhị phân của markers.

cv::aruco::drawDetectedMarkers(): Sau khi các markers đã được phát hiện, hàm này giúp vẽ lại các markers đó lên ảnh gốc để hiển thị hoặc phân tích.

Chi tiết về các hàm và input, output xem chi tiết ở link:

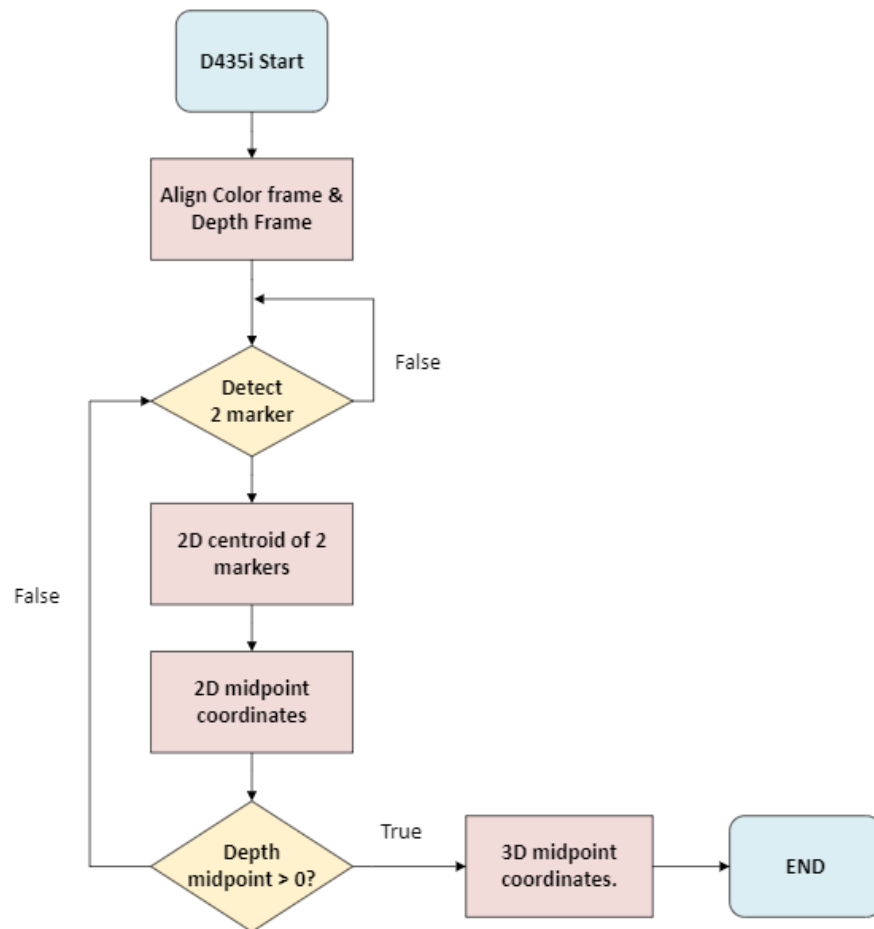
https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html



Kết quả nhận được của thuật toán

3.2 Algorithm flowchart

Tổng quan với bài toán đặt ra từ đầu, ta có sơ đồ thuật toán như sau:



Sơ đồ tổng quan bài toán

Thuật toán trên bao gồm các bước cơ bản như sau:

a. Khởi động Camera.

Trong quá trình khởi động, việc cấu hình cũng ảnh hưởng quan trọng để đảm bảo thu được kết quả tối ưu từ các dữ liệu thu thập được. Một vấn đề cần giải quyết trong quá trình này là sự khác biệt về góc nhìn (*FOV - Field of View*) giữa ảnh độ sâu và ảnh RGB.

Trong các camera RealSense, ảnh độ sâu thường có góc nhìn rộng hơn so với ảnh RGB. Điều này có thể gây ra sai số khi ta lấy chiều sâu từ ảnh Depth, nhưng lấy dữ liệu marker từ ảnh RGB. Để khắc phục điều này, quá trình căn chỉnh (aligning) giữa hai loại hình ảnh là cần thiết. Căn chỉnh đảm bảo rằng mỗi điểm trên ảnh RGB tương ứng chính xác với điểm tương ứng trên ảnh độ sâu, cho phép một sự kết hợp chính xác giữa thông tin màu và thông tin không gian của đối tượng.


```
// Căn chỉnh các khung hình
auto aligned_frames = align.process(static_cast<rs2::frameset>(frames));

rs2::depth_frame depth_frame = aligned_frames.get_depth_frame(); // lấy ảnh depth
rs2::video_frame color_frame_aligned = aligned_frames.get_color_frame(); // lấy ảnh màu
```

b. Phát hiện tọa độ tâm của 2 marker

Sau khi có hình ảnh từ Camera, quá trình tìm kiếm marker được bắt đầu. Như đã trình bày ở phần trước đó, thuật toán ArUco marker sẽ giúp tìm ra được tọa độ tâm của chúng. Sau khi ta có 2 marker, ta hoàn toàn có thể tính toán được điểm chính giữa của chúng dựa theo công thức tính trung bình.

```
if (!aligned_marker_corners.empty()) {
    for (const auto& marker : aligned_marker_corners) {
        float cX = (marker[0].x + marker[2].x) / 2.0f;
        float cY = (marker[0].y + marker[2].y) / 2.0f;
        aligned_centers.push_back(cv::Point2f(cX, cY));
    }
}
```

Một khâu then chốt trong quá trình xử lý là việc xác định độ sâu của hai marker đã phát hiện. Bước này không chỉ giúp chúng ta xác nhận rằng những marker này thực sự nằm trong phạm vi hoạt động mong muốn từ 0.2m đến 1.2m mà còn giúp loại bỏ khả năng nhiễu do nhận diện nhầm marker ở những vị trí khác xa hơn.

Việc kiểm tra độ sâu cũng giúp đánh giá mức độ chênh lệch giữa hai marker. Do 2 marker được đặt trên cùng 1 mặt phẳng, nếu chúng có sự khác biệt đáng kể về độ sâu, điều đó có thể chỉ ra rằng chúng không thuộc về cùng một đối tượng mà ta đang tìm. Trong trường hợp này, hệ thống sẽ loại bỏ kết quả hiện tại và quay lại vòng lặp tiếp tục quy trình tìm kiếm, đảm bảo rằng các dữ liệu được sử dụng để tính toán vị trí và định hướng trong không gian là chính xác nhất.

```
Midpoint: (353, 193.75)
Depth 1: 1.765
Depth 2: 2.408
Depth midpoint: 0.966
POINT 1 3D: X = -1.24853, Y = 0.206512, Z = 1.765
POINT 2 3D: X = 1.32255, Y = 0.303403, Z = 2.408
Midpoint 3D: X = -0.0763871, Y = 0.11737, Z = 0.966
```

Tọa độ Z1	Tọa độ Z2	Tọa độ Z thực tế
1.765	2.408	1

Kết quả chiều sâu nhận được và sai số thực tế

c. Ánh xạ tọa độ 3D

Để có thể di chuyển Robot một cách chính xác, chuyển đổi tọa độ từ 2D sang 3D là một bước cần thiết, vì nó cho phép chúng ta dịch chuyển thông tin từ không gian ảnh sang không gian thực của robot hoặc môi trường ảo. Để thực hiện điều này, chúng ta dựa vào ma trận Intrinsic của camera, và thành phần chiều sâu của vật. C++ cung cấp một số hàm cho trước để có thể biến đổi một cách dễ dàng:

```
rs2_deproject_pixel_to_point(point1_3d, &depth_intrin, reinterpret_cast<const float*>(&original_centers[0]), depth_1);  
rs2_deproject_pixel_to_point(point2_3d, &depth_intrin, reinterpret_cast<const float*>(&original_centers[1]), depth_2);  
rs2_deproject_pixel_to_point(midpoint_3d, &depth_intrin, reinterpret_cast<const float*>(&original_midpoint), depth_mid);
```

Thay đổi do Align: Một điểm cần lưu ý là khi thực hiện việc căn chỉnh (aligning) giữa các khung hình, ví dụ như căn chỉnh khung hình màu với khung hình độ sâu, ma trận Intrinsic bị thay đổi. Việc căn chỉnh này thường được thực hiện để đồng nhất hóa góc nhìn giữa các cảm biến khác nhau, nhưng nó cũng có thể làm thay đổi các thông số quang học được mô tả trong ma trận Intrinsic.

```
thuyle@hobo-junn:~/Desktop/main/build$ ./ArucoRealSense_ne  
Trước khi căn chỉnh:  
Width: 640  
Height: 480  
Focal length (fx): 389.142  
Focal length (fy): 389.142  
Principal point (ppx): 322.228  
Principal point (ppy): 241.031  
Distortion model: Brown Conrady  
Distortion coefficients: 0 0 0 0 0  
  
Sau khi căn chỉnh:  
Width: 640  
Height: 480  
Focal length (fx): 617.585  
Focal length (fy): 618.131  
Principal point (ppx): 324.327  
Principal point (ppy): 240.012  
Distortion model: Inverse Brown Conrady  
Distortion coefficients: 0 0 0 0 0
```

Sự thay đổi của ma trận Intrinsic

Vì tọa độ 3D phụ thuộc vào Intrinsic, do đó cần phải đưa đúng đầu vào thích hợp để đảm bảo đạt được kết quả chính xác nhất.

4. Đánh giá sai số của thuật toán

4.1 Đánh giá sai số

Đánh giá sai số					
error point 1 x	error point 1 y	error point 1 z	error point 2 x	error point 2 y	error point 3 z
-1,68533	0,3675	2,1	-1,5438	-0,0941	2
-1,67556	0,4135	2,3	-1,5438	-0,0941	2
-1,67556	0,4135	2,3	-1,5438	-0,0941	2
-1,67556	0,4135	2,3	-1,589	-0,1179	1,9
-1,68533	0,3675	2,1	-1,4986	-0,0704	2,1
-1,67556	0,4135	2,3	-1,589	-0,1179	1,9
-1,67556	0,4135	2,3	-1,5438	-0,0941	2
-1,68533	0,3675	2,1	-1,5438	-0,0941	2
-1,68533	0,3675	2,1	-1,5438	-0,0941	2
-1,68533	0,3675	2,1	-1,5754	-0,0704	2,1
-1,68533	0,3675	2,1	-1,5438	-0,0941	2
-1,68533	0,3675	2,1	-1,4986	-0,0704	2,1
-1,69022	0,2669	2	-1,6207	-0,0941	2
-1,67556	0,3361	2,3	-1,6207	-0,0941	2
-1,68533	0,3675	2,1	-1,5438	-0,0941	2
-1,68533	0,29	2,1	-1,5438	-0,0941	2
-1,681971875	0,36878125	2,16875	-1,5553875	-0,09263125	2,00625

Kết quả và sai số của thiết bị D435

Trong quá trình đánh giá sai số sử dụng thiết bị D435, ta nhận thấy sai số khoảng 1-3cm ở 2 trục X và Z, mức này khá lớn và không phù hợp cho các ứng dụng robot di động trong thực tế. Cụ thể hơn, khi sử dụng ứng dụng đo đạc Realsense-viewer từ nhà sản xuất, chiều sâu đo được là 97cm so sai số hẵn 3cm so với 100cm đo được ngoài thực tế. Điều này có thể bắt nguồn từ việc cài đặt hoặc cấu hình của D435i. Để giải quyết vấn đề này, em đã thử điều chỉnh một số cài đặt cấu hình của phần cứng thông qua lập trình và thiết lập cấu hình của nhà sản xuất để xem kết quả sai số có thay đổi hay không.

4.2 Hiệu chỉnh sai số

```
rs2::device dev = profile.get_device();
rs2::depth_sensor depth_sensor = dev.first<rs2::depth_sensor>();
depth_sensor.set_option(RS2_OPTION_VISUAL_PRESET, RS2_RS400_VISUAL_PRESET_MEDIUM_DENSITY);
```

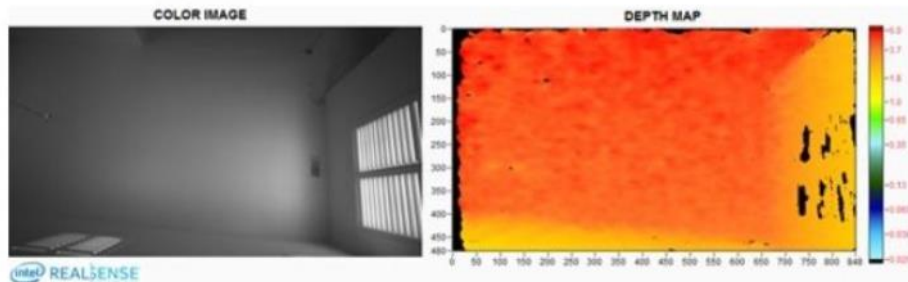
Em đã tiến hành thay đổi cấu hình của camera D435 theo các hướng dẫn được cung cấp trên trang GitHub của IntelRealSense và tài liệu chính thức về cách tối ưu hóa hiệu suất camera chiều sâu ở link

<https://github.com/IntelRealSense/librealsense/issues/2723> và <https://dev.intelrealsense.com/docs/tuning-depth-cameras-for-bestperformance#section-verify-performance-regularly-on-a-flat-wall-or-target>

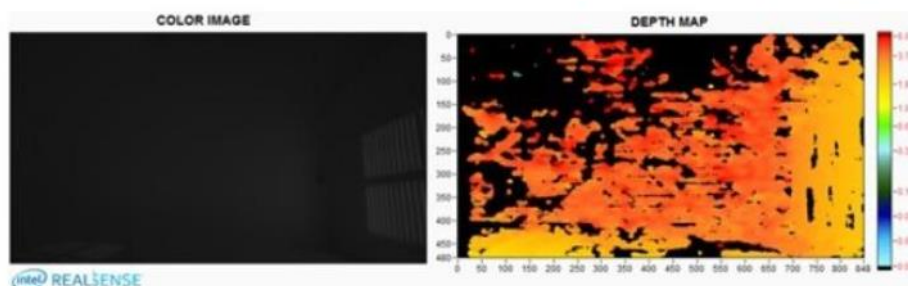


Hình minh hoạt thay đổi ROI trong Realsense Viewer

Mặc dù đã thực hiện một số điều chỉnh như thay đổi độ phân giải, điều chỉnh phơi sáng, và các thao tác khác nhằm cải thiện chất lượng hình ảnh và độ chính xác của chiều sâu, kết quả thu được vẫn chưa đạt kỳ vọng.



Example: Good



Example: Bad (under exposed)

Hình minh họa thay đổi độ sáng



Em đã thử nghiệm với việc thiết lập ROI (Region of Interest) để tập trung vào khu vực quan trọng hơn trong khung hình. Điều này giúp tối ưu hóa việc xử lý hình ảnh và

cải thiện độ chính xác của dữ liệu chiều sâu thu được từ camera. Hình minh họa sự thay đổi cấu hình cho thấy những điều chỉnh về độ phơi sáng và độ sáng, tuy nhiên, mặc dù đã có những thay đổi này, sai số 3cm vẫn tồn tại.

Những kết quả này cho thấy rằng, dù đã thay đổi khá nhiều thông số nhưng việc đo lường chiều sâu vẫn còn gặp phải những hạn chế nhất định với camera D435. Điều này chỉ ra rằng, sai số có thể ở ngay chính camera đang sử dụng. Do đó em đã đề xuất thử sử dụng một thiết bị khác để xem có chính xác lỗi ở quá trình triển khai bài toán hay lỗi ở chính phần cứng sản phẩm.

4.3 Thay đổi thiết bị D435f

Intel RealSense D435f là một phiên bản nâng cấp của dòng camera D435, được thiết kế để cung cấp dữ liệu chiều sâu chính xác hơn và ổn định hơn trong các điều kiện môi trường phức tạp. D435f được trang bị thêm các bộ lọc phần cứng bổ sung, giúp cải thiện khả năng chống nhiễu và nâng cao hiệu suất trong việc thu thập dữ liệu chiều sâu. Do đó, em sẽ sử dụng đồng thời D435 và D435f để so sánh độ chính xác của chúng.

Feature	D435/D435i	D435f/D435if
		
Use Environment	Indoor & Outdoor	Indoor & Outdoor
Image Sensor Technology	Global Shutter	Global Shutter
Depth FOV (H × V) ⁽¹⁾	87° × 58°	87° × 58°
Depth Resolution	Up to 1280 × 720	Up to 1280 × 720
Depth Accuracy ⁽²⁾	<2% at 2 m	<2% at 2 m
Depth Frame Rate	Up to 90 fps	Up to 90 fps
Depth Filter	All-Pass	IR-Pass
RGB Sensor Technology	Rolling Shutter	Rolling Shutter
RGB Resolution and Frame Rate	1920 × 1080 at 30 fps	1920 × 1080 at 30 fps
RGB Sensor FOV (H × V) ⁽¹⁾	69° × 42°	69° × 42°

Hình so sánh D435 và D435f

Điểm khác biệt: Camera độ sâu Intel RealSense D435f được thiết kế để giải quyết những thách thức đối với Robot AMR trong môi trường ánh sáng thay đổi. Camera độ sâu D435f dựa trên mẫu D435 màn trập toàn cục trường nhìn rộng phổ biến với bộ lọc

thông qua IR được gắn vào cảm biến độ sâu stereo. Bộ lọc thông qua IR làm tăng hiệu quả cường độ tương đối của mẫu máy chiếu IR có kết cấu. Điều này nâng cao chất lượng nhiễu độ sâu và phạm vi hiệu suất trong nhiều môi trường cụ thể của Robot. D435f cũng cung cấp cảm biến độ sâu màn trập toàn cục độ phân giải cao cho chuyển động tốc độ cao, rất quan trọng đối với AMR.

Nguyên lý hoạt động: Việc bổ sung bộ lọc thông IR vào D435f giúp cải thiện chất lượng độ sâu của camera stereo trong các cảnh có hoa văn dọc lặp lại, chẳng hạn như các cột ở sân bay, rèm dọc, ống trên trần nhà, cống thoát nước hoặc gạch lát sàn nhỏ. Nó cũng giúp giảm thiểu các phản xạ có thể nhìn thấy như đèn trên cao phản chiếu từ sàn nhà hoặc bàn sáng bóng. Camera D435f cũng được chế tạo để tăng hiệu suất phạm vi trong môi trường không có kết cấu trong nhà như tường trắng. Camera stereo RealSense thường không hoạt động tốt trong nhà và ngoài trời, nhưng bộ lọc thông IR giúp camera D435f lý tưởng cho ánh sáng mặt trời chói chang.

Em đã thực hiện một bài so sánh đánh giá sai số của 2 Camera khi đặt chúng đồng thời trên 1 trục như hình dưới đây:



Setup hệ thống so sánh 2 cam

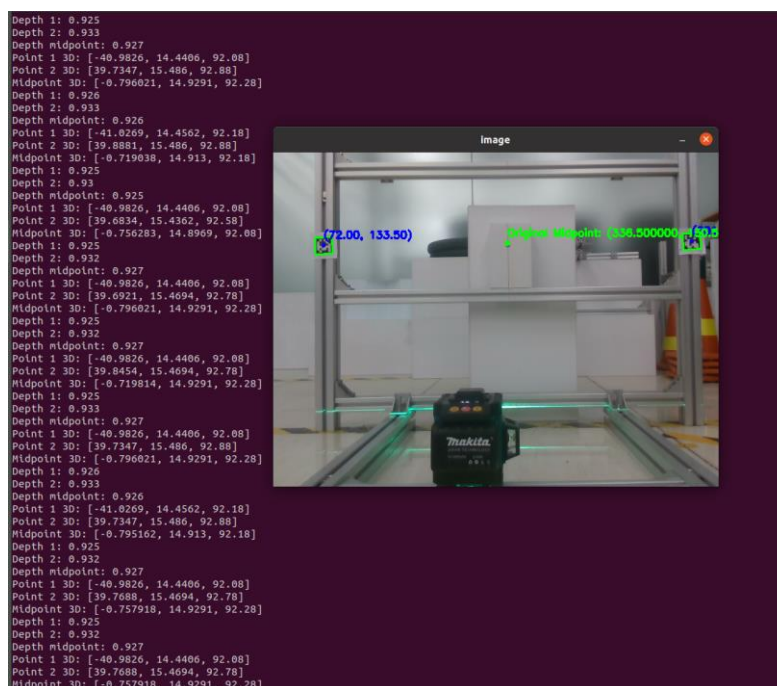
d435f (y = 17)	d435 (y = 22)
Point 1 3D: [46.7577, 28.2362, 97.5]	Point 1 3D: [-35.1265, 16.3363, 92.9]
Point 2 3D: [-31.9438, 27.5341, 101]	Point 2 3D: [41.8879, 20.1566, 94.7]
Midpoint 3D: [0, 0, 0]	Midpoint 3D: [0, 0, 0]
Aligned Midpoint 2D: (340, 138.5)	Aligned Midpoint 2D: (337.75, 158.5)
Original Midpoint 2D: (374.75, 66.5)	Original Midpoint 2D: (361, 110.25)
Depth 1: 0.975	Depth 1: 0.93
Depth 2: 1.012	Depth 2: 0.946
Depth midpoint: 0.999	Depth midpoint: 0
Point 1 3D: [-30.8369, 26.5011, 97.5]	Point 1 3D: [-35.1644, 16.3539, 93]
Point 2 3D: [48.5321, 29.3077, 101.2]	Point 2 3D: [41.8437, 20.1353, 94.6]
Midpoint 3D: [8.15638, 28.0424, 99.9]	Midpoint 3D: [0, 0, 0]
Aligned Midpoint 2D: (339.75, 138.25)	Aligned Midpoint 2D: (337.75, 158.5)
Original Midpoint 2D: (374.75, 66.5)	Original Midpoint 2D: (361, 110.25)
Depth 1: 0.975	Depth 1: 0.93
Depth 2: 1.012	Depth 2: 0.947
Depth midpoint: 0	Depth midpoint: 0
Point 1 3D: [-30.8369, 26.5011, 97.5]	Point 1 3D: [-35.1644, 16.3539, 93]
Point 2 3D: [48.5321, 29.3077, 101.2]	Point 2 3D: [41.8879, 20.1566, 94.7]
Midpoint 3D: [0, 0, 0]	Midpoint 3D: [0, 0, 0]

Giá trị trả về khi đặt 2 cam với nhau

Ta có thể thấy, cùng với 1 thuật toán, kết quả cho thấy giá trị độ sâu của vật thể khi sử dụng đồng thời D435 và D435f có sự khác biệt rõ rệt. Trong khi sai số của D435f <1cm, nhưng D435 lại ở mức 5cm:

Real	D435f	D435
98	97.5	93
Error	0.5	5

Chính vì kết quả đo đạc trên, em quyết định sử dụng camera D435f vì độ chính xác cao hơn.



Kết quả khi sử dụng D435f

Để có thể đánh giá khách quan kết quả cần phải xét đánh giá ở nhiều vị trí khác nhau của camera.

4.4 Đánh giá sai số của D435f

Từ các giá trị đánh giá sai số tại các vị trí khi dịch camera đi theo trục X 2cm, 5cm, 8cm thì ta được bảng sai số ở dưới đây. Có thể thấy rằng sai số đã giảm rất nhiều so với việc sử dụng D435 cơ bản. Sai số ở mức $< 5\text{mm}$, phù hợp trong việc tích hợp Robot trong sử dụng Robot trong công nghiệp.

X = 0								AvgX1	AvgX2
X1	Z1	X2	Z2	Error x1	error z1	error x2	error z2		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5	-0,4245970238	-0,2886970238
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,4133	92,7	36,7782	93	-0,4133	-0,2	-0,2218	-0,5		
-37,373	92,6	36,6596	92,7	-0,373	-0,1	-0,3404	-0,2		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,3326	92,5	36,7386	92,9	-0,3326	0	-0,2614	-0,4		
-37,4133	92,7	36,7782	93	-0,4133	-0,2	-0,2218	-0,5		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,4133	92,7	36,8573	93,2	-0,4133	-0,2	-0,1427	-0,7		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,373	92,6	36,6596	92,7	-0,373	-0,1	-0,3404	-0,2		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,373	92,6	36,7782	93	-0,373	-0,1	-0,2218	-0,5		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		
-37,4133	92,7	36,7386	92,9	-0,4133	-0,2	-0,2614	-0,4		

X = 8								AvgX1	AvgX2
X1	Z1	X2	Z2	error X1	error z1	error x2	error z2		
-44,7406	93,2	29,6552	93,2	0,2594	-0,7	-0,6552	-0,7	0,3670700565	0,3670700565
-44,7406	93,2	29,6552	93,2	0,2594	-0,7	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,7406	93,2	29,6552	93,2	0,2594	-0,7	-0,6552	-0,7		
-44,5966	92,9	29,6552	93,2	0,4034	-0,4	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,5916	93	0,3554	-0,5	-0,5916	-0,5		
-44,7406	93,2	29,6552	93,2	0,2594	-0,7	-0,6552	-0,7		
-44,7406	93,2	29,687	93,3	0,2594	-0,7	-0,687	-0,8		
-44,6446	93	29,687	93,3	0,3554	-0,5	-0,687	-0,8		
-44,7406	93,2	29,687	93,3	0,2594	-0,7	-0,687	-0,8		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,7406	93,2	29,687	93,3	0,2594	-0,7	-0,687	-0,8		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,6552	93,2	0,3554	-0,5	-0,6552	-0,7		
-44,6446	93	29,687	93,3	0,3554	-0,5	-0,687	-0,8		
-44,7406	93,2	29,6552	93,2	0,2594	-0,7	-0,6552	-0,7		
-44,5966	92,9	29,6552	93,2	0,4034	-0,4	-0,6552	-0,7		

Khi ta đặt ở vị trí gốc hệ ($X=0, Z = 92$), tọa độ thực của 2 marker lần lượt là $X = 37$ và -37 , giá trị nhận được so với giá trị thực sai số ở khoảng 4mm. Và khi lần lượt dịch chuyển camera theo trục X, giá trị nhận được so với thực vẫn chỉ ở dưới 5mm.

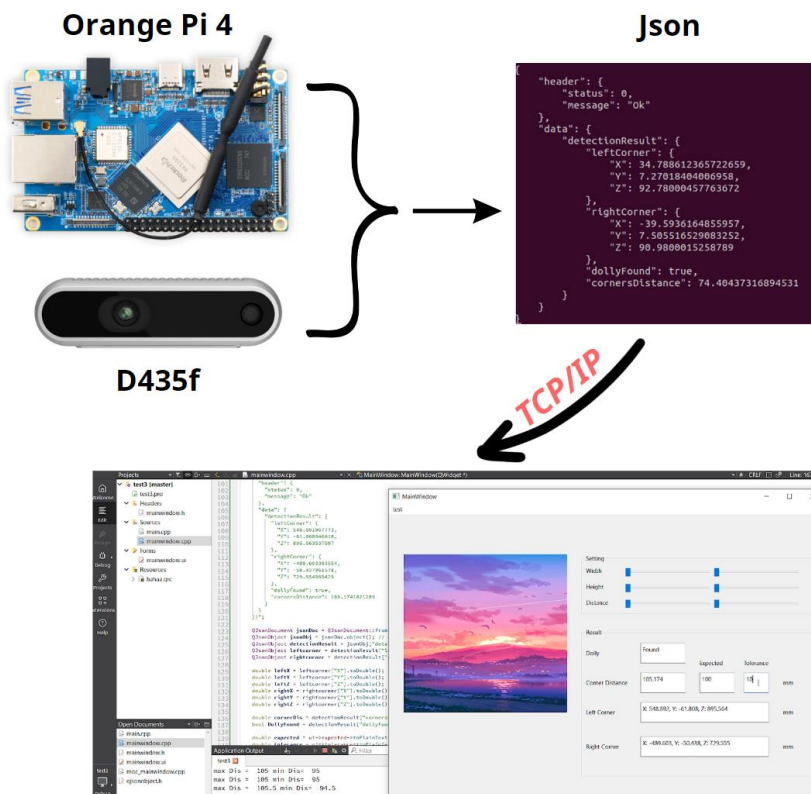
X = -2									
X1	Z1	X2	Z2	error X1	error z1	error x2	error z2	-0,525015864	0,3143807365
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,4699	92,6	38,7376	93,3	-0,4699	-0,1	0,2624	-0,8		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,4699	92,6	38,6131	93	-0,4699	-0,1	0,3869	-0,5		
-35,5082	92,7	38,5716	92,9	-0,5082	-0,2	0,4284	-0,4		
-35,4315	92,5	38,6131	93	-0,4315	0	0,3869	-0,5		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,4315	92,5	38,6131	93	-0,4315	0	0,3869	-0,5		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,5848	92,9	38,6961	93,2	-0,5848	-0,4	0,3039	-0,7		
-35,5848	92,9	38,6895	93	-0,5848	-0,4	0,3105	-0,5		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,5082	92,7	38,5716	92,9	-0,5082	-0,2	0,4284	-0,4		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,5082	92,7	38,6961	93,2	-0,5082	-0,2	0,3039	-0,7		
-35,5082	92,7	38,7376	93,3	-0,5082	-0,2	0,2624	-0,8		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
-35,5082	92,7	38,6131	93	-0,5082	-0,2	0,3869	-0,5		
X = 5									
X1	Z1	X2	Z2	Error x1	error z1	error x2	error z2	-0,1940816667	0,0744283333
-32,2244	92,9	42,1124	93,3	-0,2244	-0,4	-0,1124	-0,8	avgX1	avgX2
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,1551	92,7	42,1124	93,3	-0,2244	-0,2	-0,1124	-0,8		
-32,1551	92,7	41,977	93	-0,1551	-0,2	0,023	-0,5		
-32,1551	92,7	41,977	93	-0,1551	-0,2	0,023	-0,5		
-32,1551	92,7	42,0673	93,2	-0,1551	-0,2	-0,0673	-0,7		
-32,2244	92,9	42,0673	93,2	-0,1551	-0,4	-0,0673	-0,7		
-32,2244	92,9	42,1124	93,3	-0,2244	-0,4	-0,1124	-0,8		
-32,1551	92,7	42,0673	93,2	-0,2244	-0,2	-0,0673	-0,7		
-32,2244	92,9	42,1124	93,3	-0,1551	-0,4	-0,1124	-0,8		
-32,1551	92,7	42,0673	93,2	-0,2244	-0,2	-0,0673	-0,7		
-32,2244	92,9	42,0673	93,2	-0,1551	-0,4	-0,0673	-0,7		
-32,1551	92,7	42,1124	93,3	-0,2244	-0,2	-0,1124	-0,8		
-32,2244	92,9	42,0673	93,2	-0,1551	-0,4	-0,0673	-0,7		
-32,2244	92,9	42,1124	93,3	-0,2244	-0,4	-0,1124	-0,8		
-32,2244	92,9	42,0673	93,2	-0,2244	-0,4	-0,0673	-0,7		
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,2244	92,9	41,977	93	-0,2244	-0,4	0,023	-0,5		
-32,1551	92,7	42,1124	93,3	-0,2244	-0,2	-0,1124	-0,8		

5. Kết luận và chiến lược thực thi bài toán

Như vậy, sau quá trình thực hiện và triển khai bài toán theo 2 phương pháp: Point Cloud + RANSAC và ArUco Marker thì có thể thấy được, phương án thứ 2 là hoàn toàn khả thi hơn. Nó có thể đưa ra được các kết quả với sai số < 10mm trong nhiều trường hợp khác nhau trong thực tế. Bên cạnh đó, mặc dù đã nỗ lực khắc phục 1 số lỗi phần cứng và setup của D435 nhưng có vẻ không khả thi, do vậy có thể thấy được rằng D435f có độ chính xác cao hơn và phù hợp với bài toán đặt ra lần này hơn.

PHẦN II. ORANGE PI 4 VÀ GIAO THỨC TRUYỀN THÔNG TCP/IP

I. Sơ đồ hệ thống



Sơ đồ trên minh họa một hệ thống tích hợp Orange Pi 4 và camera Intel RealSense D435f, đóng vai trò mấu chốt trong việc nhận diện giá hàng, hỗ trợ cho robot di chuyển chính xác trong môi trường thực tế. Trong hệ thống này, Orange Pi 4 đảm nhiệm vai trò trung tâm, thu thập và xử lý dữ liệu từ camera D435f để xác định các đặc điểm không gian như vị trí và khoảng cách của các góc cạnh của giá hàng.

Để truyền tải dữ liệu một cách rõ ràng và trực quan hơn cho robot, hệ thống đã sử dụng định dạng JSON để đóng gói thông tin. Chuỗi JSON này sau đó được truyền tới giao diện người dùng (UI), nơi dữ liệu được hiển thị một cách trực quan thông qua giao thức TCP/IP.

Cụ thể quá trình được diễn ra như sau:

Step 1: Thu thập dữ liệu bằng camera D435f: Camera D435f được sử dụng để ghi nhận dữ liệu về hình ảnh và khoảng cách trong không gian ba chiều (3D). Nó có khả năng xác định vị trí các đối tượng (giá hàng) trong môi trường thực với độ chính xác cao thông qua việc đo khoảng cách từ camera đến các đối tượng và ma trận Intrinsics của nó. Thông qua đó, ta có thể ghi nhận được các thông tin như:

tọa độ 3D 2 điểm marker, khoảng cách tới marker, khoảng cách của 2 marker với nhau, góc lệch.... Những dữ liệu này, đủ để Robot có thể đưa ra những quyết định di chuyển phù hợp với mục đích của mình.

Step 2: Xử lý dữ liệu trên Orange Pi 4: Để D435f có thể thực hiện nhiệm vụ của mình, cần phải sử dụng một kit nhúng - Orange Pi 4, một máy tính nhỏ gọn và mạnh mẽ. Từ đó D435f sẽ tính toán và xác định tọa độ X, Y, Z của các điểm quan trọng như các góc trái và phải của đối tượng.

Step 3: Định dạng dữ liệu thành JSON: Sau khi xử lý và lấy được thông tin từ D435f, Orange Pi 4 sẽ đóng gói thông tin này dưới dạng JSON (JavaScript Object Notation), một định dạng dữ liệu nhẹ và dễ dàng trao đổi giữa các hệ thống. Trong JSON này, bao gồm các tọa độ của các điểm, khoảng cách giữa các góc, và trạng thái phát hiện (dollyFound) của đối tượng.

Step 4: Truyền dữ liệu qua TCP/IP: TCP/IP là một giao thức mạng chuẩn, cho phép truyền thông tin một cách ổn định và hiệu quả giữa các thiết bị. Dữ liệu JSON sau đó được truyền qua giao thức TCP/IP từ Pi đến một máy tính hoặc thiết bị khác để hiển thị kết quả một cách trực quan hơn.

Step 5: Hiển thị kết quả trên giao diện người dùng (UI): Cuối cùng, dữ liệu JSON được nhận và hiển thị trên giao diện người dùng (UI) được phát triển trong QT Creator. Giao diện này cung cấp các thông tin trực quan như khoảng cách giữa các góc, vị trí các góc trong không gian, và trạng thái phát hiện của đối tượng. Người dùng có thể theo dõi và tương tác với các thông số này trực tiếp trên giao diện, do đó có thể giám sát, phân tích dữ liệu, thậm chí là điều khiển đưa ra các lệnh phù hợp với các tiền kết quả.

II. Orange Pi 4

Trong các phần trên, việc xử lý dữ liệu camera D435/D435f được thực hiện toàn toàn trên máy tính cá nhân. Tuy nhiên để tích hợp trong Robot ARM, thì cần chuyển sang sử dụng một nền tảng nhúng như Orange Pi 4. Orange Pi sẽ đóng vai trò chính, thay thế máy tính cá nhân để nạp, chạy code và điều khiển robot. Nó sẽ làm công việc điều khiển các cảm biến, xử lý dữ liệu từ camera trong thời gian thực, định hướng di chuyển và xác định chính xác vị trí của mình trong không gian. Nhờ khả năng xử lý mạnh mẽ, kết nối linh hoạt và kích thước nhỏ gọn, Orange Pi 4 trở thành bộ não điều khiển toàn bộ hệ thống robot. Điều này giúp giảm sự phụ thuộc vào máy tính cá nhân, tạo ra một hệ thống di động, gọn nhẹ, tối ưu cho các ứng dụng công nghiệp.

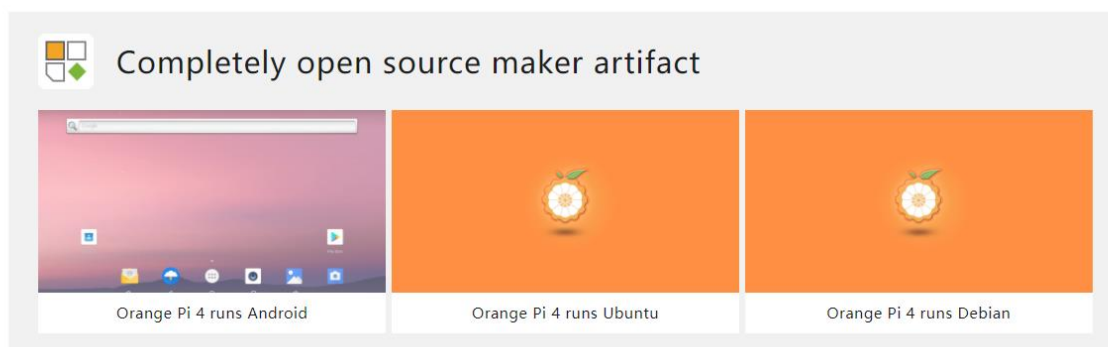


Hình ảnh Orange Pi 4

CPU	<ul style="list-style-type: none"> • Rockchip RK3399 (28nm HKMG process) • 6-core ARM® 64-bit processor ,main frequency speeds up to 2.0GHz • Based on the large and small size core architecture of big.LITTLE: • Dual-core Cortex-A72 (large core) + Quad-core Cortex-A53 (small core)
GPU	<ul style="list-style-type: none"> • Mali-T864 GPU • Supports OpenGL ES1.1/2.0/3.0/3.1, OpenVG1.1,OpenCL, DX11, support for AFBC
PMU	RK808 PMU
Memory+Onboard Storage	<ul style="list-style-type: none"> • Dual 4GB LPDDR4 + 16GB EMMC Flash • Dual 4GB LPDDR4 +EMMC Flash(Default Empty)
On-board WiFi+Bluetooth	AP6256, IEEE 802.11 a/b/g/n/ac, BT5.0
Network	10/100/1000Mbps Ethernet(Realtek RTL8211E)
Audio	<ul style="list-style-type: none"> • Output: 3.5mm Jack and HDMI2.0a • Input: MIC
Video Outputs	<ul style="list-style-type: none"> • 1 x HDMI 2.0 (Type-A), Supports 4K@60fps output • 1 x DP 1.2 (Display Port) , Supports 4K@60fps output • Supports Dual MIPI-DSI (4 lines per channel)
Camera	2 x MIPI-CSI Camera connector (MIPI_RX0、MIPI_TX1/RX1)
USB	2 x USB2.0 HOST, 1x USB3.0 HOST, 1 x USB3.0 Type-C
RTC	Support RTC, on-board battery backup interface
Debug UART	3 pins Debug UART
GPIO	<ul style="list-style-type: none"> • GPIO1 40 pins (1 x I2S、2 x I2C、1 x SPI/UART、8 x GPIO) • GPIO2 24pin PCIE port

Bảng thông số của Orange Pi 4

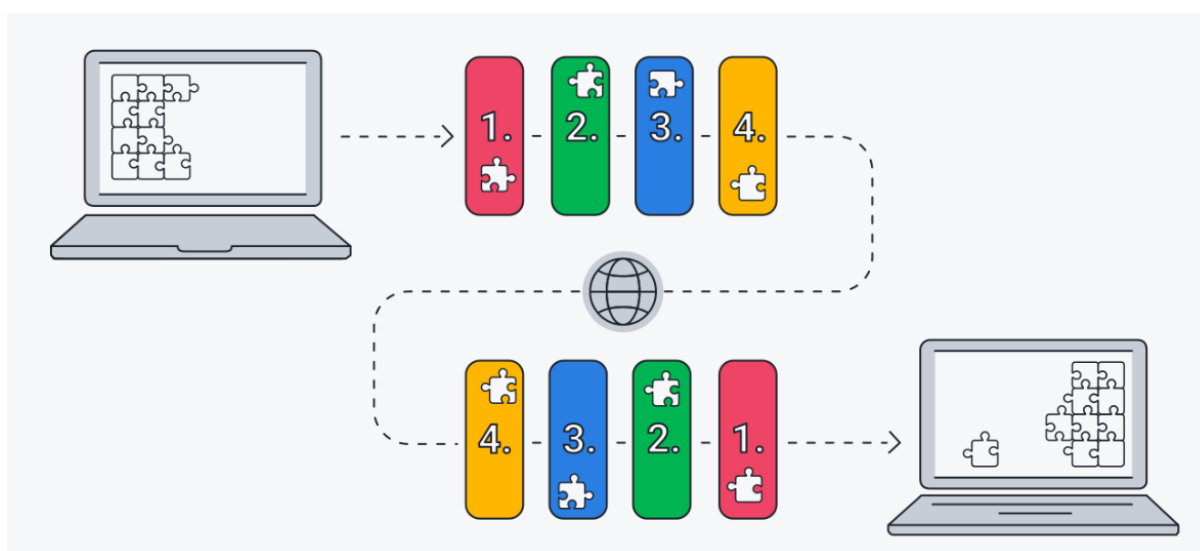
Orange Pi được chạy trên nền Ubuntu, nên việc xử lý lấy thông tin từ camera D435f gần như không có sự khác biệt so với trước đây trên laptop cá nhân.



III. Giao thức TCP/IP

1. Nguyên lý hoạt động

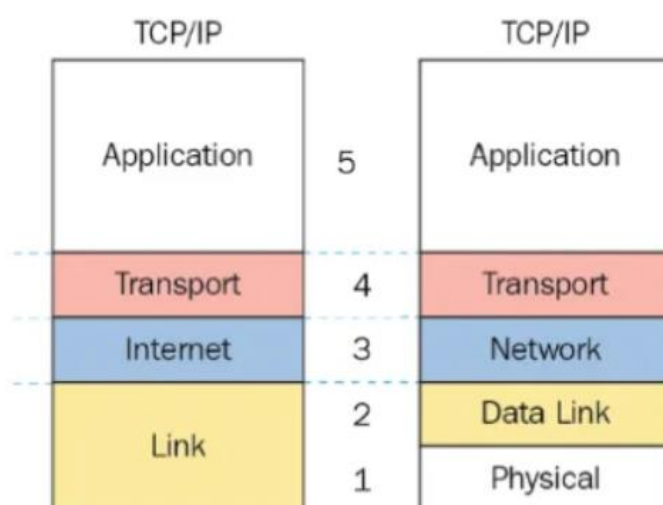
TCP/IP là một giao thức liên kết dữ liệu được sử dụng trên internet để cho phép máy tính và các thiết bị khác gửi và nhận dữ liệu. TCP/IP là viết tắt của Transmission Control Protocol/Internet Protocol và giúp các thiết bị được kết nối với internet có thể giao tiếp với nhau qua mạng. Giao thức này đảm bảo rằng dữ liệu được truyền đi một cách đáng tin cậy từ thiết bị gửi đến thiết bị nhận, đồng thời xác định cách dữ liệu sẽ được định tuyến và xử lý trong quá trình truyền tải.



Hình minh họa TCP/IP

Nguyên lý hoạt động của TCP/IP dựa trên một loạt các bước và tầng (layers) để đảm bảo dữ liệu được truyền tải từ một thiết bị này đến một thiết bị khác qua mạng một cách an toàn, đáng tin cậy, và có hiệu quả. Bộ giao thức TCP/IP được chia thành bốn tầng

chính, mỗi tầng đảm nhiệm một vai trò cụ thể trong quá trình truyền dữ liệu.



Tầng Ứng dụng (Application Layer)

- **Chức năng:** Đây là tầng cao nhất, nơi các ứng dụng người dùng tương tác với giao thức mạng. Các giao thức tầng ứng dụng như HTTP, FTP, SMTP, và Telnet hoạt động ở tầng này, cung cấp các dịch vụ như truyền tệp, duyệt web, gửi email,...

Tầng Giao vận (Transport Layer)

- **Chức năng:** Tầng này chịu trách nhiệm đảm bảo rằng dữ liệu được truyền giữa hai thiết bị một cách đáng tin cậy và theo thứ tự chính xác. TCP (Transmission Control Protocol) và UDP (User Datagram Protocol) là hai giao thức chính trong tầng này.
 - **TCP:** Dữ liệu từ tầng ứng dụng được chia thành các gói nhỏ hơn (segments). TCP đảm bảo rằng các gói này đến đích theo đúng thứ tự và không bị mất mát, thông qua cơ chế bắt tay ba bước (three-way handshake), đánh số thứ tự các gói, và xác nhận nhận gói (ACK).
 - **UDP:** Ngược lại, UDP là giao thức không kết nối (connectionless) và không đảm bảo sự toàn vẹn của gói dữ liệu, nhưng nhanh hơn, phù hợp cho các ứng dụng yêu cầu tốc độ cao như streaming video hoặc game online.

Tầng Mạng (Network Layer)

- **Chức năng:** Tầng mạng chịu trách nhiệm định tuyến các gói dữ liệu từ nguồn đến đích qua mạng. Giao thức chính trong tầng này là IP (Internet Protocol), với

các phiên bản IPv4 và IPv6.

- **IP:** Mỗi gói dữ liệu được gán địa chỉ IP của nguồn và đích. Tầng mạng sử dụng thông tin này để xác định đường đi tốt nhất qua các router và mạng khác nhau để đưa gói dữ liệu đến đích cuối cùng.
- **ICMP (Internet Control Message Protocol):** Một phần của tầng mạng, ICMP được sử dụng để gửi thông điệp lỗi và các thông báo liên quan đến tình trạng của mạng (như ping).

Tầng Liên kết Dữ liệu và Vật lý (Data Link and Physical Layer)

- **Chức năng:** Tầng này xử lý việc truyền dữ liệu qua phương tiện vật lý như cáp mạng, sóng radio, hoặc các loại kết nối khác. Nó cũng bao gồm việc đóng gói dữ liệu vào các khung (frames) và điều khiển truy cập vào phương tiện truyền tải.
 - **Ethernet/Wi-Fi:** Dữ liệu được truyền dưới dạng các khung (frames) trên các phương tiện truyền tải vật lý như cáp Ethernet hoặc mạng không dây Wi-Fi. Các khung này chứa địa chỉ MAC, một địa chỉ duy nhất của mỗi thiết bị mạng, giúp điều phối việc truyền dữ liệu giữa các thiết bị trong cùng một mạng vật lý.

2. Triển khai hệ thống

Để gửi dữ liệu từ camera cho Robot, ở đây em sử dụng chuỗi JSON. RapidJSON là một thư viện phân tích JSON hiệu quả cao cho C++, có tốc độ phân tích nhanh. Nó còn tiêu thụ ít tài nguyên hệ thống, làm cho nó trở thành lựa chọn thích hợp cho các ứng dụng cần phản hồi nhanh và tiết kiệm tài nguyên. Với RapidJSON, người dùng có thể xử lý lượng lớn dữ liệu từ cảm biến hoặc dữ liệu điều khiển mà không làm ảnh hưởng đến hiệu suất chung của hệ thống robot.


```
[
  {
    "header": {
      "status": 0,
      "message": "Ok"
    },
    "data": {
      "detectionResult": {
        "leftCorner": {
          "X": 34.788612365722659,
          "Y": 7.27018404006958,
          "Z": 92.78000457763672
        },
        "rightCorner": {
          "X": -39.5936164855957,
          "Y": 7.505516529083252,
          "Z": 90.9800015258789
        },
        "dollyFound": true,
        "cornersDistance": 74.40437316894531
      }
    }
  }
]
```

Trong hệ thống này Orange Pi sẽ đóng vai trò là *client* thu thập dữ liệu và xử lý dữ liệu từ cảm biến, sau đó sẽ gửi dữ liệu dạng JSON lên server thông qua địa chỉ IP của laptop (đóng vai trò *server*) để gửi dữ liệu này qua giao thức TCP/IP.

```
# Thiết lập kết nối đến server
server_ip = '192.168.1.100' # Địa chỉ IP của server
server_port = 12345 # Cổng của server

# Tạo socket
client_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

try:
    # Kết nối tới server
    client_socket.connect((server_ip, server_port))

    # Gửi dữ liệu JSON
    client_socket.sendall(json_data.encode('utf-8'))

    # Đóng kết nối
    client_socket.close()
```

Bên cạnh đó, phía Server, cũng phải thực hiện việc tiếp nhận thông tin qua cổng Port để đẩy dữ liệu lên giao diện UI trên QT


```
server_ip = '0.0.0.0' # Lắng nghe trên tất cả các địa chỉ IP
server_port = 12345

# Tạo socket
server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Gán địa chỉ IP và cổng cho socket
server_socket.bind((server_ip, server_port))

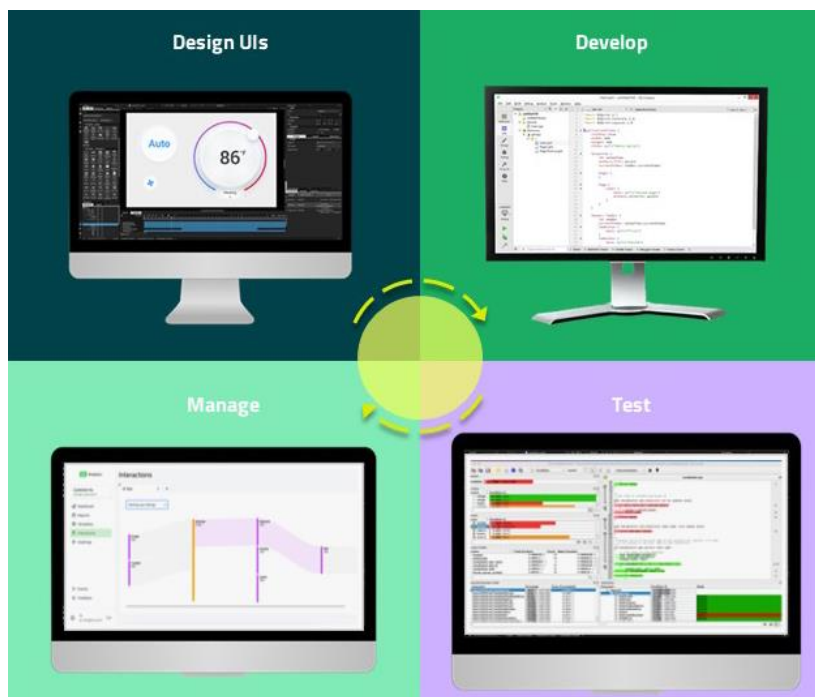
# Lắng nghe kết nối
server_socket.listen(5)
print(f"Server listening on {server_ip}:{server_port}")
```

Kết thúc quá trình, Server sẽ nhận được dữ liệu là một loạt các chuỗi JSON như mong muốn. Và chuỗi JSON này sẽ được tiếp tục đẩy lên trên giao diện người dùng để có thể trực quan hóa hơn.

PHẦN III. THIẾT KẾ GIAO DIỆN THÔNG QUA QT CREATOR

I. Giới thiệu tổng quan về QT Creator




QT Creator là một bộ công cụ toàn diện không chỉ giúp xây dựng, biên dịch, gỡ lỗi, và triển khai các ứng dụng đa nền tảng (cross-platform) dựa trên các hệ điều hành như Windows, macOS, Linux, Android và iOS.



Các tính năng của QT demo

Một số tính năng nổi bật của QT Creator bao gồm:

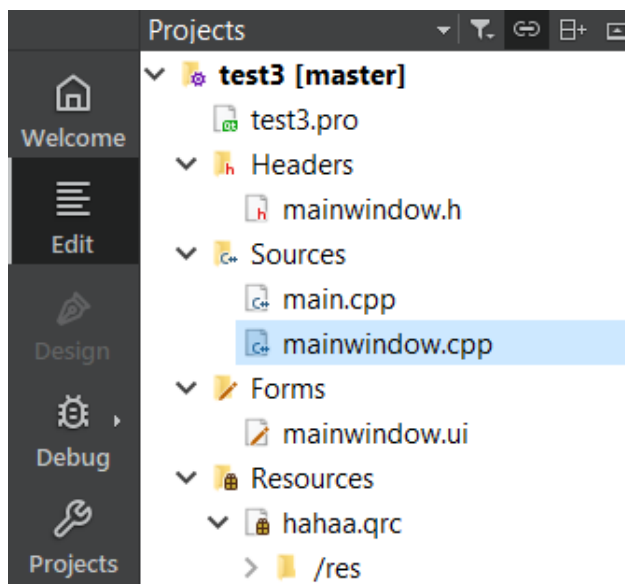
1. **Hỗ trợ mã nguồn mạnh mẽ:** QT Creator hỗ trợ nhiều ngôn ngữ lập trình khác nhau, nhưng chủ yếu tập trung vào C++ và QML (ngôn ngữ lập trình sử dụng trong Qt).
2. **Thiết kế giao diện người dùng (UI):** QT Creator có công cụ thiết kế giao diện trực quan (Qt Designer), cho phép kéo thả các thành phần UI và tạo ra các giao diện người dùng phức tạp mà không cần viết mã bằng tay.
3. **Gỡ lỗi và biên dịch:** IDE này tích hợp sẵn các công cụ gỡ lỗi (debugging) và biên dịch (compilation) mạnh mẽ, giúp phát hiện lỗi và tối ưu hóa mã nguồn.
4. **Phát triển đa nền tảng:** QT Creator hỗ trợ phát triển ứng dụng trên nhiều nền tảng khác nhau từ cùng một mã nguồn, giúp tiết kiệm thời gian và công sức khi cần triển khai ứng dụng trên nhiều hệ điều hành.

		
Work with Code	Build and Run Applications	Use Tools and Extensions
<ul style="list-style-type: none"> › Edit Code › Debug › Analyze › Test 	<ul style="list-style-type: none"> › Manage Projects › Manage Kits › Build and Run › Develop for Devices 	<ul style="list-style-type: none"> › Use Qt Creator › Use the UI › Design UIs › Create Models and Diagrams › Read Documentation

Minh họa các tính năng nổi bật của QT

II. Cấu trúc dự án

Cấu trúc của dự án bao gồm các file như sau:



Đầu tiên là file **.pro**: Đây là tệp tin dự án chính (project file) của QT, định nghĩa các thông số cần thiết cho quá trình biên dịch và liên kết. Nó bao gồm thông tin về các tệp nguồn, thư viện cần liên kết, và các cài đặt cấu hình khác.

```

1 //QT += core gui
2 QT += core gui network
3 greaterThan(QT_MAJOR_VERSION, 4): QT += widgets
4
5 CONFIG += c++17
6
7 SOURCES += \
8     main.cpp \
9     mainwindow.cpp
10
11 HEADERS += \
12     mainwindow.h
13
14 FORMS += \
15     mainwindow.ui
16
17 # Default rules for deployment.
18 qnx: target.path = /tmp/${TARGET}/bin
19 else: unix:!android: target.path = /opt/${TARGET}/bin
20 !isEmpty(target.path): INSTALLS += target
21
22 RESOURCES += \
23     hahaa.qrc

```

Ví dụ như ở trên đã chỉ ra các mô-đun core, gui, network dự án sử dụng và chuẩn C++17 trong việc lập trình ngôn ngữ. Nó liệt kê các tệp Source, tệp header, và UI, cùng với các quy tắc triển khai ứng dụng trên các hệ điều hành khác nhau.

Tiếp theo, **Header**. Thư mục này thường chứa các tệp tin header (.h), nơi khai báo các lớp, hàm, và biến được sử dụng trong chương trình. Header files giúp tách biệt giữa phần khai báo và phần định nghĩa của mã nguồn, giúp mã dễ quản lý hơn.

```

3
4 #include <QMainWindow>
5 #include <QTcpSocket>
6 #include <QByteArray>
7 #include <QTcpServer>
8 #include <QJsonObject>
9
10 QT_BEGIN_NAMESPACE
11 namespace Ui {
12     class MainWindow;
13 }
14 QT_END_NAMESPACE
15
16 class MainWindow : public QMainWindow
17 {
18     Q_OBJECT
19
20 public:
21     MainWindow(QWidget *parent = nullptr);
22     ~MainWindow();
23
24 private slots:
25     void onSliderValueChanged();
26     void on_expected_textChanged();
27     void on_tolerance_textChanged();
28
29 private:
30     Ui::MainWindow *ui;
31     QImage originalPixmap;
32     QTcpSocket *socket;
33     QTcpServer *server;
34     QJsonObject currentJsonObj;
35     void cropImage(int width1, int width2, int height1, int height2);
36     void startServer();
37     void updateUIFromJson(const QJsonObject &jsonObj);
38     void onNewConnection();
39     void onReadyRead();
40 };
41 #endif // MAINWINDOW_H
42

```

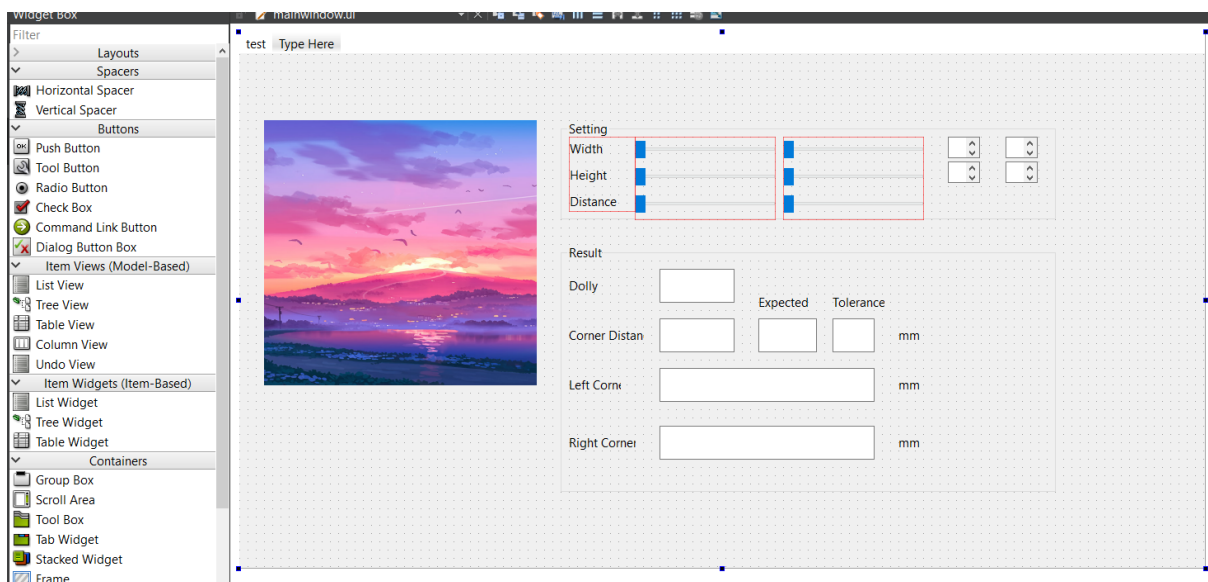
Sources: là thư mục chứa các tệp tin mã nguồn chính (.cpp), nơi các hàm và lớp đã được khai báo trong header file. Thông thường sẽ có hai tệp tin chính là main.cpp và mainwindow.cpp.

- **main.cpp:** Đây thường là nơi chương trình bắt đầu thực thi. Nó thường chứa hàm main() và khởi tạo các thành phần cốt lõi của ứng dụng, bao gồm cả đối tượng QApplication.
- **mainwindow.cpp:** Đây chính là phần code chính của người dùng. Nó bao gồm việc xử lý các sự kiện giao diện người dùng, vẽ giao diện, và quản lý các widget trong cửa sổ chính.

```
connect(ui->expected, &QTextEdit::textChanged, this, &MainWindow::on_expected_textChanged);  
connect(ui->tolerance, &QTextEdit::textChanged, this, &MainWindow::on_tolerance_textChanged);
```

Ví dụ như ở đây, ta hoàn toàn có thể đưa các con số đầu vào bất kỳ từ người dùng trên giao diện *textChanged*, và kết quả đầu ra sẽ phải thông qua các con số đó để chọn lọc kết quả phù hợp.

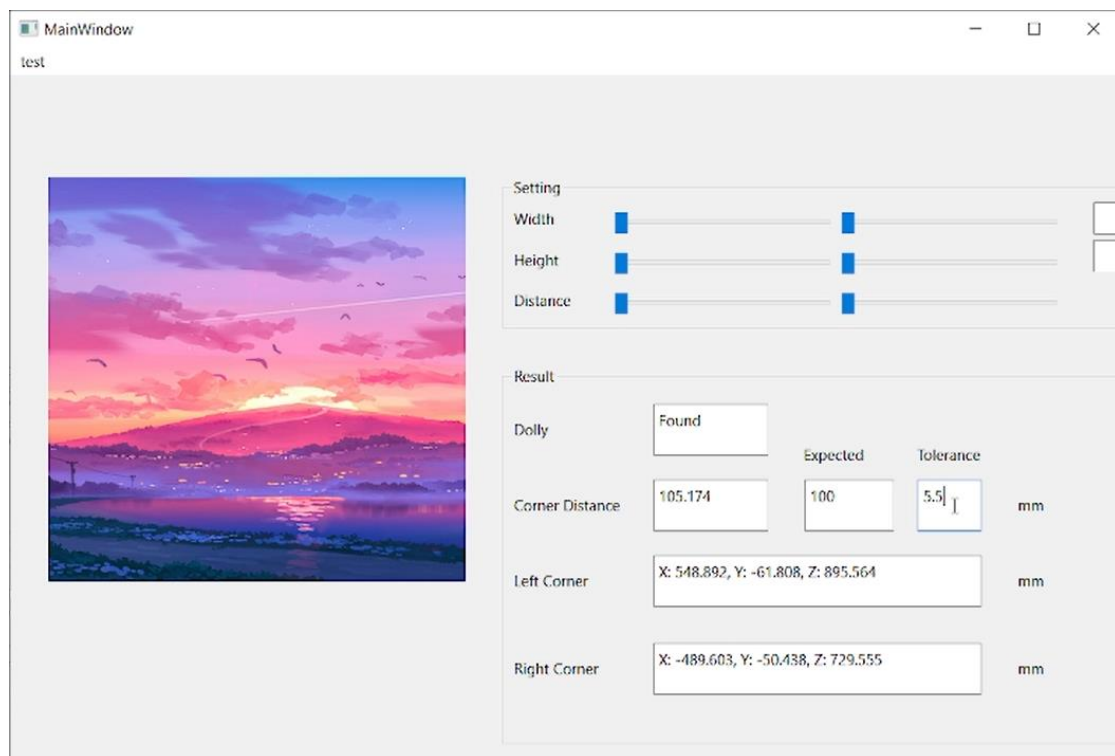
Forms: Thư mục này chứa các tệp giao diện (.ui) được thiết kế bằng công cụ thiết kế UI của Qt. Những tệp này xác định cấu trúc và bố cục của giao diện ứng dụng, bao gồm các widget, layout, và các kết nối tín hiệu-slot giữa chúng.



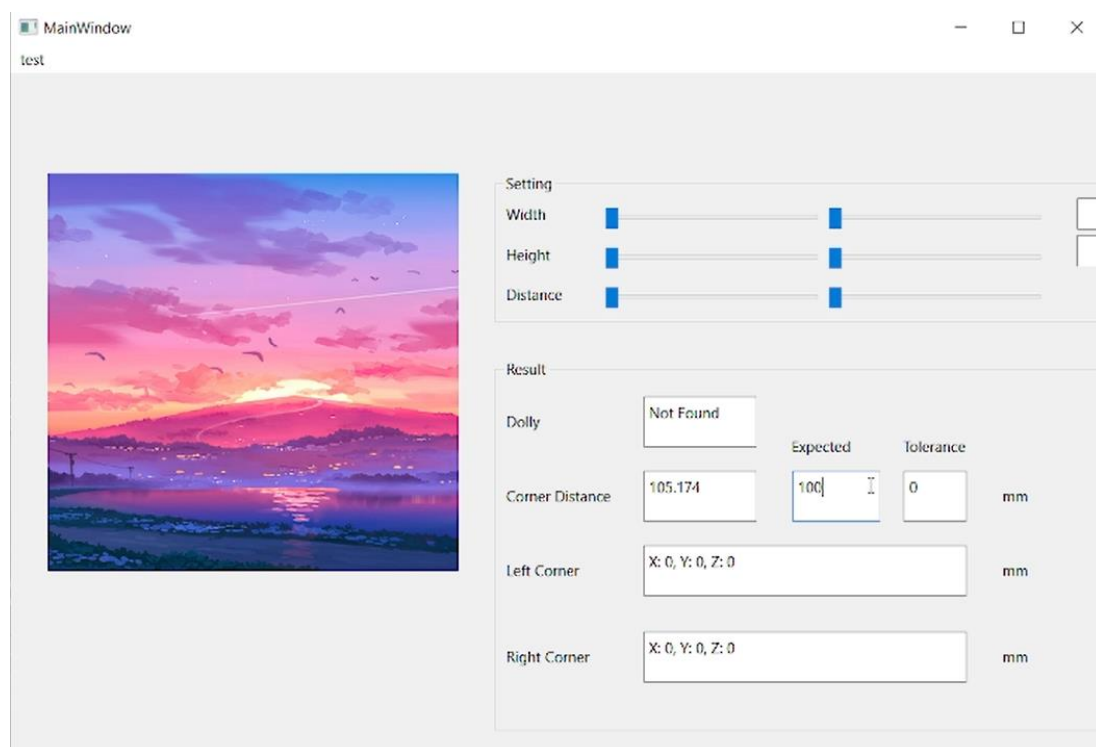
Resources: Thư mục này thường chứa các tệp tin tài nguyên như hình ảnh, biểu tượng, hoặc các tệp văn bản mà ứng dụng cần sử dụng.

Như vậy QT Creator giúp quản lý tất cả các thành phần này một cách trực quan, cho phép người dùng phát triển chức năng của ứng dụng. IDE này cung cấp các công cụ mạnh mẽ như trình biên dịch, trình gỡ lỗi, và tích hợp hệ thống kiểm soát phiên bản, giúp quy trình phát triển phần mềm trở nên hiệu quả hơn.

III. Tích hợp hệ thống



Trên là hình minh họa giao diện khi nhận được đầu vào là các thông số từ Orange Pi đẩy lên. Các giá trị này sẽ được chọn lọc tiếp thông qua 2 giá trị **Expected** và **Tolerance** được nhập vào từ người dùng. Và khi các giá trị đó thay đổi thì kết quả hiển thị trên UI cũng sẽ thay đổi theo:



PHẦN IV. KẾT LUẬN

TASK1 đã hoàn thành được các công việc như sau: Lấy thông tin nhận diện giá hàng và convert từ tọa độ ảnh sang tọa độ thực. Phần sau đã đánh giá thêm được tọa độ của camera khi đặt so với tọa độ của xe và các sai số đi kèm.

Các phần chưa đạt được: giao diện QT vẫn còn khá đơn giản và chưa fix được xong lỗi truyền ảnh/video realtime từ camera đẩy lên (lỗi mất ảnh, dữ liệu). Và vì lý do con pi die giữa chừng nên chưa thể test được sau khi nén ảnh/video gửi thông tin qua và giải nén nó liệu có ổn không (đọc thêm truyền thông đa phương tiện).

Một trong những cách giải pháp khác của task1 chính là task2, không sử dụng aruco marker nữa mà sử dụng vision/AI/yolo.....