

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐẠI HỌC SƯ PHẠM KỸ THUẬT
THÀNH PHỐ HỒ CHÍ MINH



CÔNG TRÌNH NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN

TÁI CẤU TRÚC VẬT THỂ 3D TỪ CẢP HÌNH ẢNH
STEREO CAMERA

MÃ SỐ: SV2021-125



Tp. Hồ Chí Minh, tháng 10/2021

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TPHCM
❧ ❁ ❁



BÁO CÁO TỔNG KẾT
ĐỀ TÀI NGHIÊN CỨU KHOA HỌC CỦA SINH VIÊN
**TÁI CẤU TRÚC VẬT THỂ 3D TỪ CẤP
HÌNH ẢNH STEREO CAMERA**
SV2021-125

Chủ nhiệm đề tài: Nguyễn Tân Lực

TP Hồ Chí Minh, 10/2021

BỘ GIÁO DỤC VÀ ĐÀO TẠO
TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TPHCM



BÁO CÁO TỔNG KẾT

ĐỀ TÀI NGHIÊN CỨU KHOA HỌC SINH VIÊN

TÁI CẤU TRÚC VẬT THỂ 3D TỪ CẶP HÌNH ẢNH STEREO CAMERA

SV2021-125

Chủ nhiệm đề tài: Nguyễn Tân Lực

Thuộc nhóm ngành khoa học: Kỹ thuật

SV thực hiện: Nguyễn Tân Lực Nam, Nữ: Nam

Dân tộc: Kinh

Lớp, khoa: 18151CL3B, khoa Đào tạo Chất lượng cao

Năm thứ: 3

Số năm đào tạo: 4

Ngành học: Công nghệ kỹ thuật điều khiển và tự động hóa

Người hướng dẫn: PGS.TS Lê Mỹ Hà

TP Hồ Chí Minh, 10/2021

MỤC LỤC

DANH MỤC HÌNH ẢNH	3
DANH MỤC NHỮNG TỪ VIẾT TẮT	6
THÔNG TIN KẾT QUẢ NGHIÊN CỨU CỦA ĐỀ TÀI	7
MỞ ĐẦU	9
CHƯƠNG 1: TỔNG QUAN	11
1.1. Phương pháp nghiên cứu.....	11
1.2. Nguyên lý phương pháp tái cấu trúc vật thể 3D	11
1.3. Các mô hình biến thể kỹ thuật trong phương pháp tái cấu trúc vật thể 3D	11
1.3.1. Trong nước.....	12
1.3.2. Ngoài nước.....	12
1.4. Nội dung của đề tài	17
CHƯƠNG 2: CƠ SỞ LÝ THUYẾT.....	18
2.1. Tìm hiểu về stereo camera	18
2.1.1. Mô hình pinhole camera	18
2.1.2. Lens distortion	19
2.1.3. Homogeneous Coordinates	21
2.2. Cân bằng sáng (histogram equalization).....	22
2.3. Gaussian blur.....	23
2.4. Hiệu chỉnh máy ảnh (camera calibration)	25
2.5. Hiệu chỉnh stereo camera	27
2.6. Geometry of image formation	28
2.7. Stereo matching:.....	33
2.7.1. Tổng quan:	33
2.7.2. Phương pháp Semi-global matching:	34

2.8. Bộ lọc trung vị (Median filter)	36
CHƯƠNG 3: XÂY DỰNG MÔ HÌNH 3D VẬT THỂ.....	38
3.1. Khối lấy dữ liệu ảnh từ camera	39
3.2. Khối hiệu chỉnh	40
3.3. Khối tính toán độ sâu ánh.....	41
3.4. Khối hiển thị mô hình 3D.....	42
CHƯƠNG 4: CHƯƠNG TRÌNH VÀ HƯỚNG DẪN SỬ DỤNG	44
4.1. Chương trình “captures.py”	44
4.2. Chương trình “Calibration_Rectification.py”	45
4.3. Chương trình “sgm_disparity.py”	46
4.4. Chương trình “pointCloud_visualize.py”	46
CHƯƠNG 5: THỰC NGHIỆM VÀ KẾT QUẢ THU ĐƯỢC	48
5.1. Thực nghiệm 1: Kết quả tốt nhất nhóm đạt được.	48
5.2. Thực nghiệm 2: Kết quả cùng phương pháp, cùng độ sáng nhưng vật thể khác.	51
5.3. Thực nghiệm 3: Kết quả cùng phương pháp, khác độ sáng, khác vật thể.	52
5.4. Thực nghiệm 4: Kết quả cùng phương pháp, khác độ sáng, khác vật thể.	52
5.5. Thực nghiệm 5: Kết quả so sánh phương pháp tìm bản đồ chênh lệch (disparity maps):	53
KẾT LUẬN VÀ KIẾN NGHỊ.....	55
TÀI LIỆU THAM KHẢO	56
PHỤ LỤC	58

DANH MỤC HÌNH ẢNH

Hình 1: Stereo camera.....	10
Hình 2 Tạo mô hình 3D từ nhiều ảnh khác nhau.....	13
Hình 3 Kết quả chất lượng tái cấu trúc 3D	13
Hình 4 (a) Cặp ảnh trái phải ,(b) Đám mây điểm từ 1 cặp,	14
Hình 5 Kết quả từ PMVS.....	14
Hình 6 Mesh model.....	14
Hình 7 Cặp hình ảnh trái phải từ stereo camera.	15
Hình 8 Mô hình 3D vật thể	15
Hình 9 Kết quả công trình của Christian Teutsch, Dirk Berndt, Andreas Sobotta, Silvio Sperling	16
Hình 10 Cặp ảnh left và right theo nghiên cứu của Josef Bigun	16
Hình 11 Kết quả của việc tìm điểm tương đồng và ước tính, táo tại mô hình 3D mặt người	16
Hình 12 Ảnh trái	17
Hình 13 Ảnh phải.....	17
Hình 14 Kết quả độ sáng chỉ độ sâu	17
Hình 15: Mô hình pinhole.....	18
Hình 16 Mô hình Pinhole.....	19
Hình 17 Radial distortion.....	20
Hình 18 Sự méo dạng tiếp tuyến	20
Hình 19	21
Hình 20 Ảnh chưa cân bằng	23
Hình 21 Ảnh đã cân bằng sáng	23
Hình 22 Đồ thị ảnh chưa cân bằng sáng	23
Hình 23 Đồ thị ảnh đã cân bằng sáng	23

Hình 24	Ảnh trước và sau khi dùng bộ lọc Gaussian blur.....	24
Hình 25	Điểm Q = (X, Y, Z) chiếu trên mặt phẳng ảnh.....	26
Hình 26	Hiệu chỉnh stereo camera.....	28
Hình 27	Toạ độ trong không gian.....	28
Hình 28	Chuyển đổi từ kích thước sang toạ độ pixel	30
Hình 29	Toạ độ điểm principal	30
Hình 30	Phép chiếu toạ độ 3D đến toạ độ 3D	32
Hình 31	Biến đổi census.....	33
Hình 32	Dữ liệu sau biến đổi census.	34
Hình 33	Hamming Distance.	34
<i>Hình 34</i>	<i>Phương pháp semi-global matching.....</i>	35
Hình 35	Ví dụ về hoạt động của bộ lọc trung vị.....	37
Hình 36	Ảnh gốc.....	37
Hình 37	Ảnh đã được lọc.....	37
Hình 38	Sơ đồ khói của quá trình xây dựng mô hình 3D.....	38
Hình 39	Lưu đồ thu thập dữ liệu từ stereo camera	39
Hình 40	Các loại méo dạng ảnh.....	40
Hình 41	Disparity trước khi lọc trung vị	42
Hình 42	Disparity sau khi lọc trung vị.....	42
Hình 43	Ví dụ về mô hình 3D dạng .ply.....	43
Hình 44	Sơ đồ khói mối liên hệ giữa các chương trình	44
Hình 45	Hệ thống kết nối stereo camera với máy tính	45
Hình 46	Chạy chương trình Calibration Rectification.....	45
Hình 47	Quá trình tính toán disparity	46
Hình 48	Chạy chương trình pointCloud_visualize.py	47

Hình 49 Giao diện cửa sổ để xem mô hình 3D của vật thể.....	47
Hình 50 Ảnh trái	48
Hình 51 Ảnh phải.....	48
Hình 52 So sánh ảnh sau khi hiệu chỉnh.....	49
Hình 53 Ảnh đã cân bằng sáng	49
Hình 54 Disparity trước khi lọc trung vị	49
Hình 55 Disparity thể hiện độ sâu ảnh.....	50
Hình 56 Mô hình 3D của vật thể	50
Hình 57 Ảnh disparity thực nghiệm 2	51
Hình 58 Đám mây điểm ảnh thực nghiệm 2	51
Hình 59 Disparity vật thể thực nghiệm 3.....	52
Hình 60 Đám mây điểm ảnh thực nghiệm 3	52
Hình 61 Disparity vật thể thực nghiệm 4.....	52
Hình 62 Đám mây điểm ảnh thực nghiệm 4	53
Hình 63 Ảnh cones trái.	53
Hình 64 Ảnh cones phải.....	53
Hình 65 Groud truth.....	53
Hình 66 Phương pháp Block matching opencv	54
Hình 67 Disparity phương pháp SSD stereo matching.....	54
Hình 68 Disparity phương pháp semi-global matching.....	54

DANH MỤC NHỮNG TỪ VIẾT TẮT

2D	2-Dimension (Không gian 2 chiều)
3D	3-Dimension (Không gian 3 chiều)
CNC	Computer Numerical Control
HD	High Definition
NCC	Normalized Cross Correlation
RGB	“red, green, blue” - đỏ, xanh lục và xanh lam
SAD	Sum of Absolute Differences
SSD	Sum of Squared Differences
SGM	Semi-global matching
USB	Universal Serial Bus (một chuẩn kết nối tuần tự đa dụng trong máy tính)
USD	United States dollar (đồng đô la Mỹ)
VR	Virtual Reality (Thực tế ảo)

BỘ GIÁO DỤC VÀ ĐÀO TẠO

TRƯỜNG ĐH SƯ PHẠM KỸ THUẬT TPHCM

THÔNG TIN KẾT QUẢ NGHIÊN CỨU CỦA ĐỀ TÀI

1. Thông tin chung:

Stt	Họ và tên	MSSV	Lớp	Khoa
1	Nguyễn Thanh Nhã	18151098	18151CL3A	Đào tạo CLC
2	Phan Thanh Truyền	18151139	18151CL2B	Đào tạo CLC

- Người hướng dẫn: PGS.TS Lê Mỹ Hà

2. Mục tiêu đề tài:

- Tìm hiểu nguyên lý cấu tạo và xây dựng mô hình Stereo camera từ hai camera logitech C310 HD.
 - Tạo mô hình 3D từ cặp hình ảnh stereo camera.
 - Tính toán khoảng cách từ vật thể tới camera.

3. Tính mới và sáng tạo:

- Áp dụng các giải thuật mới trong việc tính toán bản đồ chênh lệch.

4. Kết quả nghiên cứu:

- Kết quả thu được đạt khoảng 90% so với mục tiêu đề ra.
 - Point cloud mô tả vật thể tương đối rõ ràng và có thể nhận biết khoảng cách trước sau, có màu sắc rõ ràng như thực tế.

5. Đóng góp về mặt giáo dục và đào tạo, kinh tế - xã hội, an ninh, quốc phòng và khả năng áp dụng của đề tài:

- Áp dụng trong lĩnh vực robot để tìm thông tin và trích xuất vị trí của các vật thể 3D trong không gian thực.
 - Ứng dụng phù hợp cho các hệ thống tự hành.
 - Tái tạo không gian ảo 3D.
 - Ứng dụng trong thực tế ảo (VR).

6. Công bố khoa học của SV từ kết quả nghiên cứu của đề tài: Không

Ngày 10 tháng 10 năm 2021
SV chịu trách nhiệm chính
thực hiện đề tài
(kí, họ và tên)

X

Luc Nguyen Tan
Student

Nhận xét của người hướng dẫn về những đóng góp khoa học của SV thực hiện đề tài:

Ngày 10 tháng 10 năm 2021
Người hướng dẫn
(kí, họ và tên)

X

Ha Le My
Teacher

MỞ ĐẦU

Tái cấu trúc vật thể 3D có ý nghĩa rất lớn trong nhiều lĩnh vực của cuộc sống và các ngành khoa học kỹ thuật như hệ thống tự hành, công nghệ in nỗi 3D, công nghệ thời trang, y học, xây dựng tái tạo các di sản văn hoá, khảo cổ, ... là một trong những khâu quan trọng trong công nghiệp tự động hoá. Ở Việt Nam hiện nay, chủ đề tái tạo 3D vẫn còn mới và đang trong quá trình nghiên cứu phát triển. Ở các trường đại học, có nhiều đề tài đã từng nghiên cứu về tạo dựng ảnh 3D từ một cặp hay nhiều ảnh 2D ghép lại với nhau, từ đó thu được dữ liệu đám mây điểm, tái tạo lại hình 3D của đối tượng. Từ đó, tạo mô hình 3D của đối tượng có thể áp dụng trong giảng dạy thay vì phải hướng dẫn trực tiếp trên vật thể với số lượng ít. Đối với doanh nghiệp, nhà máy, nền sản xuất công nghiệp cơ khí đang phát triển đặc biệt là công nghệ gia công trên máy CNC dựa vào việc xác định khối vật thể 3D có thể phục vụ được nhiều ngành công nghiệp như: sản xuất ô tô, xe máy, gia công chi tiết, công nghệ khuôn mẫu, ... Với các doanh nghiệp trong nước để đầu tư vài trăm nghìn USD cho một thiết bị có thể tạo dựng mô hình 3D chính xác là khá khó khăn. Hơn nữa thiết bị nhập khẩu thì tính năng kỹ thuật không được khai thác hết do phụ thuộc vào phần mềm của hãng cũng cung cấp, bảo dưỡng sửa chữa cũng khó khăn về mặt kỹ thuật lẫn kinh tế. Việc nghiên cứu tìm hiểu loại thiết bị đo này giúp cho sử dụng hiệu quả hơn và có khả năng tự chế tạo tại Việt Nam từ đó cho phép ứng dụng rộng rãi, nâng cao chất lượng cũng như sự phát triển của ngành tự động hóa sản xuất. Để làm được điều đó, việc nghiên cứu thu thập được mô hình 3D về hình dáng, kích thước chính xác của đối tượng từ nhiều góc nhìn và các thuật toán xử lý dữ liệu rất quan trọng. Từ những yêu cầu thực tế, nhóm đã lựa chọn một phần nhỏ để xử lý tái cấu trúc vật thể 3D từ một góc nhìn với bộ hai camera (stereo vision) và lựa chọn đề tài “TÁI CẤU TRÚC VẬT THỂ 3D TỪ CẶP HÌNH ẢNH STEREO CAMERA”

Với mục tiêu:

- Tìm hiểu nguyên lý cấu tạo và xây dựng mô hình Stereo camera từ hai camera Logitech C310 HD.
- Tạo mô hình 3D từ cặp hình ảnh stereo camera.
- Tính toán khoảng cách từ vật thể tới camera.

Thông qua việc tìm hiểu cơ sở lý thuyết, nhóm đã tiến hành thực nghiệm bằng mô hình gồm cả phần cứng và phần mềm, trong đó phần cứng bao gồm 2 camera Logitech

C310 HD (Hình 1). Do thời gian thực hiện đề tài có hạn nên nhóm xin giới hạn lại phạm vi nghiên cứu của nhóm là chỉ dùng 2 camera chụp các đối tượng để tạo ra mô hình 3D gồm các đám mây điểm ảnh mô tả tương đối chiều sâu, màu sắc của vật thể trong không gian 3 chiều.



Hình 1: Stereo camera

Chương 1: TỔNG QUAN

Chương này trình bày chi tiết hơn về tổng quan dụng ảnh của vật thể trong không gian ba chiều. Trong thực tế, có nhiều cách để tạo dựng mô hình vật thể trong không gian ba chiều như: ghép nhiều ảnh 2D, từ cắp ảnh, ... Đồng thời, nguyên lý phương pháp tái cấu trúc, các mô hình trong và ngoài nước cũng được nhóm đề cập đến.

1.1. Phương pháp nghiên cứu

Phương pháp nghiên cứu: quá trình nghiên cứu được nhóm chia thành các giai đoạn. Đầu tiên nhóm tiến hành tìm hiểu tổng quan về đề tài và cơ sở lý thuyết thông qua các công cụ tìm kiếm như Google, Google scholar, Libgen, các diễn đàn tạp chí khoa học quốc tế. Bước 2, nhóm xây dựng mô hình Stereo camera từ 2 camera dựa vào cấu trúc Stereo camera do các nhà nghiên cứu phát triển đã phát minh ra sản phẩm này. Bước 3, nhóm bắt đầu xây dựng chương trình máy tính bằng việc sử dụng công cụ hỗ trợ lập trình PyCharm và sử dụng ngôn ngữ lập trình Python. Quá trình xây dựng chương trình tái thiết 3D được nhóm tham khảo từ các trang mạng phổ biến như GitHub, ở đó có các chương trình của các tác giả trước đã thực hiện và họ đã chia sẻ nó cho cộng đồng nghiên cứu. Cuối cùng, nhóm tiến hành đánh giá chất lượng và hiệu quả của chương trình bằng cách thực hiện tái thiết tại các điều kiện môi trường như ánh sáng, nhiệt độ... hoặc sử dụng các đối tượng, vật thể khác nhau.

1.2. Nguyên lý phương pháp tái cấu trúc vật thể 3D

Thiết lập hai camera tương đương như cặp mắt người. Hình ảnh 2D thu được từ camera trái và camera phải dùng để tìm chiều sâu của các đối tượng được chụp. Sau đó thông tin về chiều sâu đối tượng được kết hợp với tọa độ ảnh 2D và phương pháp biến đổi hình học 2D sang 3D sẽ thu được mạng lưới điểm ảnh trong không gian 3D (point cloud).

1.3. Các mô hình biến thể kỹ thuật trong phương pháp tái cấu trúc vật thể 3D

Từ việc sử dụng các công cụ tìm kiếm, nhóm đã tiến hành tìm hiểu tổng quan về các đề tài thị giác nổi và các công trình trong thời gian gần đây để nắm bắt được quá trình tiếp cận và kết quả đạt được của công trình đã nghiên cứu trong và ngoài nước.

1.3.1. Trong nước

Ở Việt Nam, chủ đề tái tạo 3D vẫn còn mới, và đang trong quá trình nghiên cứu phát triển

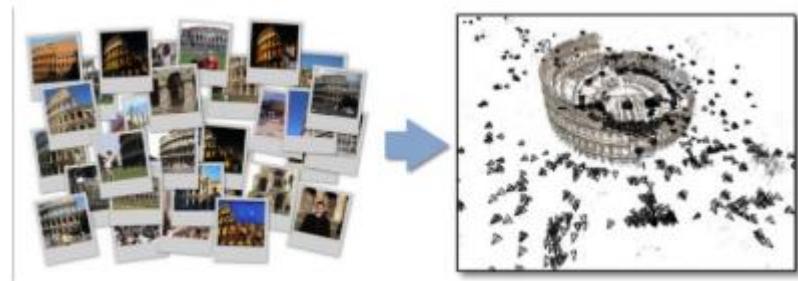
Computer Vision: Tái tạo ảnh 3D (tác giả: Phan Thị Ngát, Lớp ĐT2 K56, năm 2015). Bài viết nói về phương pháp tái tạo ảnh 3D từ một cặp ảnh 2D và xây dựng lại hệ thống cho phép tái tạo ảnh trực tiếp sử dụng một cặp camera. Từ đó ta có thể sử dụng 2 hay nhiều ảnh 2D được chụp từ cùng một đối tượng và cùng khoảng cách để tìm khoảng cách từ vật đến camera (depth map) và thu thập dữ liệu đám mây điểm, tái tạo lại hình ảnh 3D của đối tượng đó.

Nghiên cứu đo biên dạng 3D của chi tiết bằng phương pháp sử dụng ánh sáng cấu trúc (tác giả: Lê Quang Trà, chuyên ngành kỹ thuật cơ khí, Trường Đại Học Bách Khoa Hà Nội, năm 2016). Bài viết nói về phương pháp đo sử dụng ánh sáng cấu trúc dựa trên nguyên lý tam giác lượng trong quang học, ứng dụng vào đo lường biên dạng 3D các chi tiết cơ khí. Biên dạng 3D của chi tiết làm biến dạng hình ảnh mẫu chiều và được nhận biết thông qua hệ thống camera. Phân tích dữ liệu ảnh và kết hợp phương pháp mã hóa ảnh chiều để dựng lại tọa độ đám điểm của chi tiết đo. Từ đó làm chủ công nghệ đo, xây dựng cơ sở tính toán thiết kế, chế tạo thiết bị đo phù hợp với điều kiện chế tạo tại Việt Nam.

1.3.2. Ngoài nước

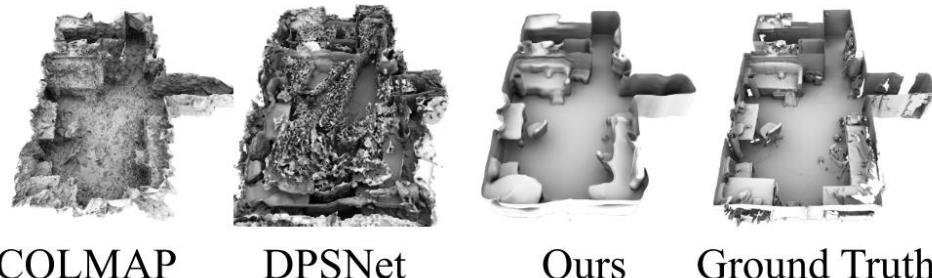
Công nghệ kĩ thuật phát triển nhanh, đặc biệt ở các nước phát triển, họ đã và đang nghiên cứu sâu vào các ứng dụng tái tạo 3D trong nhiều lĩnh vực, tái tạo các thiết bị cơ khí, y tế, quân đội, giáo dục... với độ chính xác cao, xây dựng bản đồ 3D của một khu vực ...

3D Reconstruction from Multiple Images [1] (năm 2008) (tác giả: Shawn McCann). Bài viết nói về việc phương pháp sử dụng các thuật toán “Structure from Motion và Multiview Stereo” để xây dựng mô hình 3D của các đối tượng, tòa nhà và cảnh.



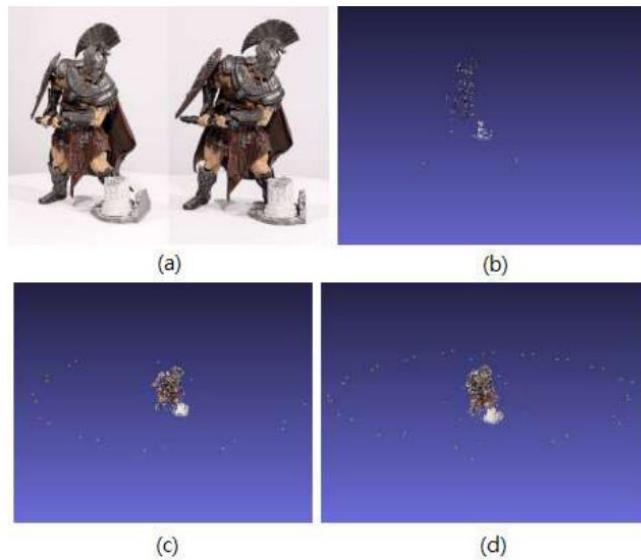
Hình 2 Tạo mô hình 3D từ nhiều ảnh khác nhau

Atlas: End-to-End 3D Scene Reconstruction from Posed Images [2] (năm 2020) (tác giả: Zak Murez, Tarrence van As, James Bartolozzi, Ayan Sinha, Vijay Badrinarayanan, and Andrew Rabinovich). Bài báo trình bày phương pháp tái tạo 3D bằng cách trực tiếp hồi quy một hàm khoảng cách có dấu (truncated signed distance function) từ một tập hợp các hình ảnh RGB. Các cách tiếp cận truyền thống để tái tạo 3D dựa trên bản đồ độ sâu, trước khi ước tính mô hình 3D đầy đủ của một cảnh, hay một đối tượng.

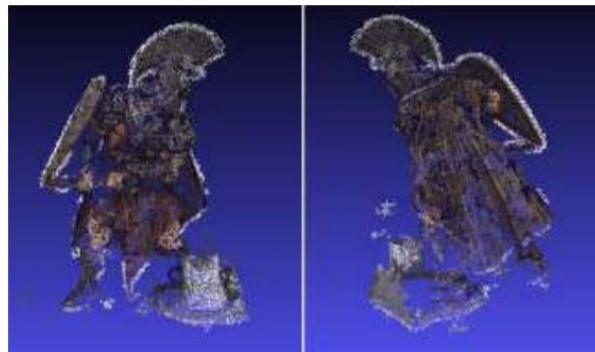


Hình 3 Kết quả chất lượng tái cấu trúc 3D

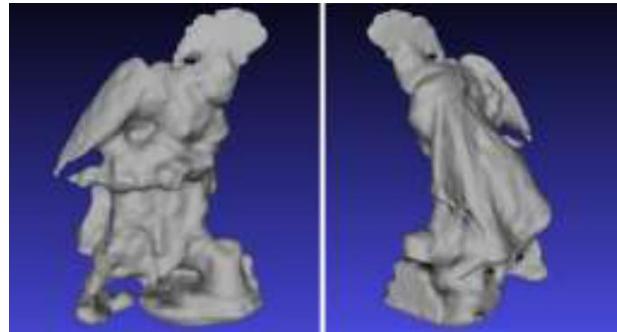
3D Model Reconstruction Based on Multiple View Image Capture [3] (năm 2012) (tác giả: Po-Han Lee, Jui-Wen Huang, Huei-Yung Lin, Department of Electrical Engineering National Chung Cheng University 168 University Road, Min-Hsiung, Chia-Yi 621, Taiwan). Trong bài báo này, tác giả đã thiết kế tái tạo một mô hình thử nghiệm. Hệ thống sử dụng máy ảnh để chụp nhiều hình ảnh từ đối tượng và phân tích những hình ảnh đó sử dụng cấu trúc từ chuyển động (Structure From Motion) để có được thông số camera và đếm mây điểm ba chiều.



Hình 4 (a) Cặp ảnh trái phải, (b) Đám mây điểm từ 1 cặp, (c) Đám mây điểm từ 16 ảnh, (d) Đám mây điểm từ 32 ảnh



Hình 5 Kết quả từ PMVS



Hình 6 Mesh model

Point-Based 3D Reconstruction of Thin Objects [4] (năm 2013) (nhóm tác giả: Benjamin Ummenhofer and Thomas Brox Computer Vision Group University of Freiburg, Germany). Bài báo nói về phương pháp tái tạo 3D hình dạng của một đối tượng từ một tập hợp các hình ảnh. Đó là phương pháp tái tạo dựa trên điểm dày đặc có thể áp dụng đối với đối tượng đặc biệt như vật thể có hình dạng mỏng. Các vật thể mỏng hầu như không có thể tích, đặt ra một thách thức đặc biệt cho việc tái tạo liên quan đến việc biểu hình dạng và tổng hợp dữ liệu chiều sâu. Tối ưu hóa bản đồ độ sâu bằng cách

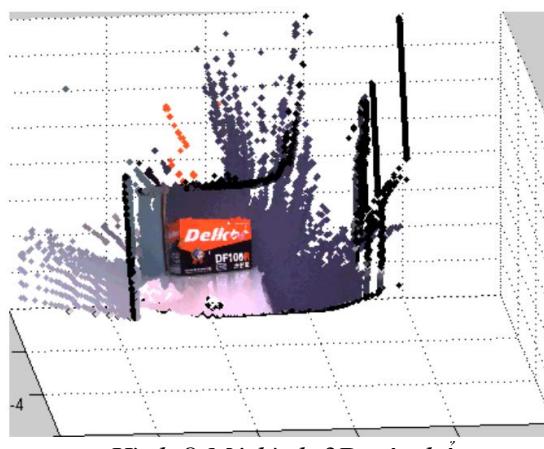
xem mỗi pixel là một điểm trong không gian. Nhóm các điểm thành super pixel và một cách tiếp cận băm không gian cho các truy vấn vùng lân cận nhanh.

3D Reconstruction and Measurement of Indoor Object Using Stereo Camera

[5] (năm 2011) của Wei-wei Ma, My-Ha Le, Kang-Hyun Jo. Bài báo này thực hiện một nghiên cứu về tái tạo 3D và đo lường các vật thể trong nhà bằng stereo camera. Đầu tiên, tính năng SIFT được trích xuất từ cả hai hình ảnh và tìm thấy các điểm phù hợp. Phương pháp RANSAC được áp dụng để loại bỏ các điểm khớp sai. Thứ hai, hai hình ảnh từ stereo camera được chỉnh sửa và tạo độ chênh lệch. Cuối cùng, bản đồ độ sâu và thông tin 3D của mỗi pixel được bắt nguồn. Phương pháp này được áp dụng để tái tạo và đo đặc vật thể thật trong môi trường trong nhà. Kết quả như sau:

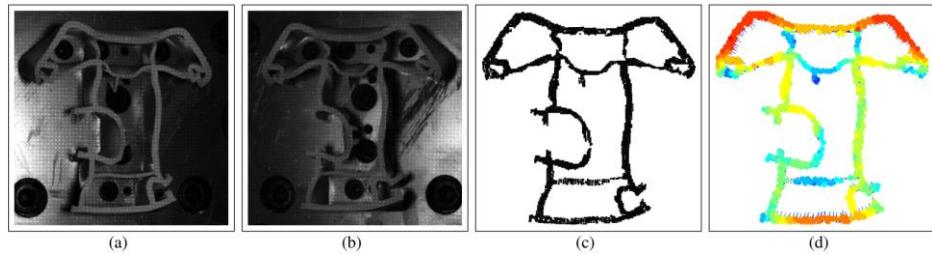


Hình 7 Cặp hình ảnh trái phải từ stereo camera.



Hình 8 Mô hình 3D vật thể

A Flexible Photogrammetric Stereo Vision System for Capturing the 3D Shape of Extruded Profiles [6] (năm 2006) của Christian Teutsch, Dirk Berndt, Andreas Sobotta, Silvio Sperling dùng 2 camera quan sát máy phun nhựa để ước lượng độ dày của khuôn nhựa có đồng đều không. Tác giả dùng phương pháp phân tích một tập hợp hàm tương quan, kích thước và hình dạng cửa sổ, dùng hệ số tương quan Pearson để đạt được một sự phù hợp tốt nhất từ cặp ảnh stereo trái phải. Kết quả như sau:



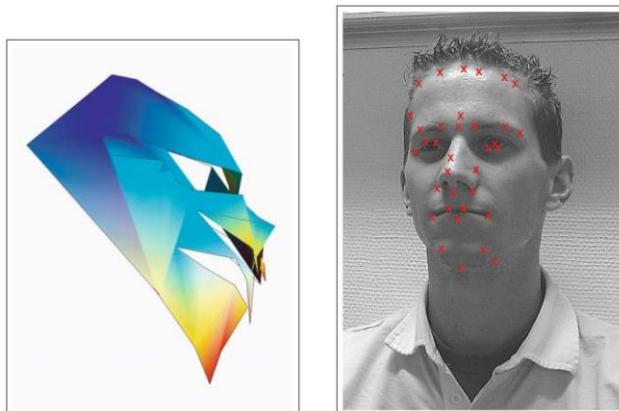
Hình 9 Kết quả công trình của Christian Teutsch, Dirk Berndt, Andreas Sobotta, Silvio Sperling

Với hình (a): ảnh trái; hình (b): ảnh phải; hình (c): ảnh disparity; hình (d) ảnh màu biểu thị độ sâu.

Vision with Direction [7] (năm 2006) của Josef Bigun tìm những điểm tương đồng ở ảnh trái và phải để xây dựng nên ảnh 3D, vấn đề này sẽ được tác giả ứng dụng trong luận văn của mình. Kết quả của Josef Bigun như sau:



Hình 10 Cặp ảnh left và right theo nghiên cứu của Josef Bigun



Hình 11 Kết quả của việc tìm điểm tương đồng và ước tính, táo tại mô hình 3D mặt người

Segment-Based Stereo Matching Using Belief Propagation and a Self-Adapting Dissimilarity Measure [1] (năm 2006) của Andreas Klaus, Mario Sormann, Konrad Karner dùng thuật toán lan truyền tin cậy và tự thích nghi sai lệch để làm phù

hợp ảnh nổi có tỉ lệ tương đồng cao, tác giả đã kiểm chứng thuật toán này trong luận văn của mình.



Hình 12 Ảnh trái



Hình 13 Ảnh phải



Hình 14 Kết quả độ sáng chỉ độ sâu

1.4. Nội dung của đề tài

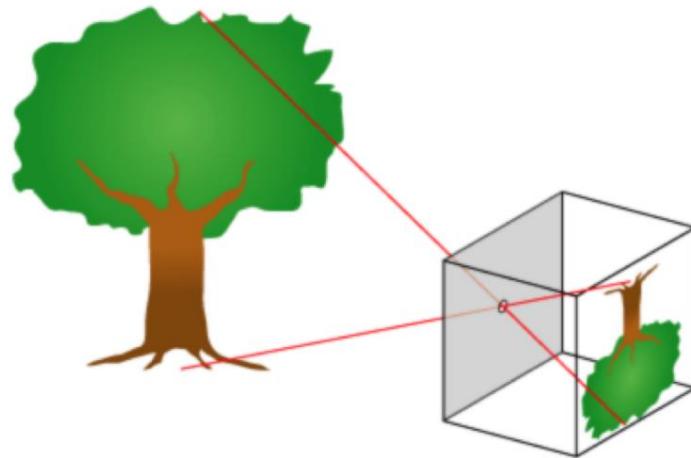
Đề tài được nhóm chia thành 6 chương, chương 1 đó là tổng quan về đề tài đã được nhóm trình bày phần trên. Chương 2 là các cơ sở lý thuyết xử lý ảnh cơ bản cần phải thành thạo vì là nền tảng để xây dựng những thuật toán phức tạp hơn, hiểu về thị giác nổi, mô hình máy ảnh lỗ kim, đồng nhất tọa độ, hình học epipolar, tất cả các tài liệu nghiên cứu về thị giác nổi đều nói về nó. Chương 3 là chương xây dựng mô hình 3D gồm camera, vật thể, khung cảnh và sơ đồ khối cho từng chức năng, từng nhiệm vụ của những cơ sở lý thuyết được nêu ở trên. Chương 4 đề cập tới chương trình code và cách để sử dụng code. Cuối cùng là chương 5 thu được kết quả từ thực nghiệm là tất cả những thí nghiệm được thể hiện đầy đủ trong đề tài mà nhóm thực hiện được, phần code của chương trình được đính kèm trong phần phụ lục.

Chương 2: CƠ SỞ LÝ THUYẾT

2.1. Tìm hiểu về stereo camera

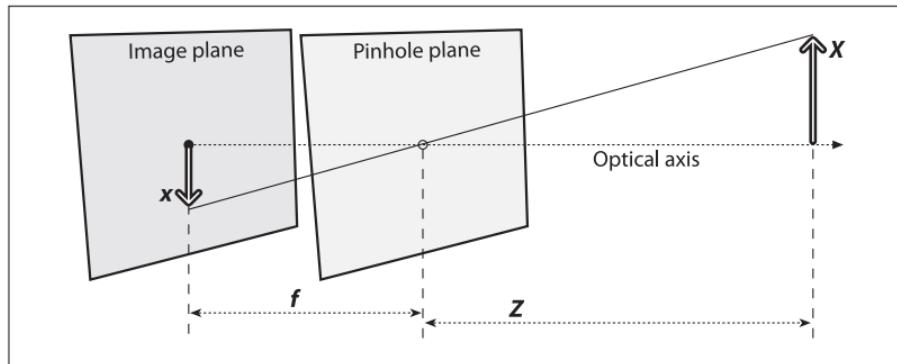
2.1.1. Mô hình pinhole camera

Mô hình pinhole camera [8] là một mô hình máy ảnh đơn giản mà không có lens và chỉ cần một lỗ nhỏ (hay có khẩu độ nhỏ), màn chắn và hộp đen. Cơ chế chụp ảnh của mô hình này là dựa vào các tia sáng phản chiếu từ đối tượng trong không gian, các tia sáng này sẽ đi qua lỗ trên hộp đen và cuối cùng nằm trên màn chắn, điều này được thể hiện Hình 15. Vì điểm sáng thu được chỉ tới từ một vài tia sáng chiếu từ vật thể trong không gian đi qua lỗ của hộp đen tiến tới màn chắn. Nên khi khẩu độ nhỏ, tương đương lượng ánh sáng đi qua lỗ ít và cho ảnh có sự sắc nét hơn, nhưng khi máy ảnh hoạt động trong điều kiện thiếu sáng thì ảnh thu được sẽ bị tối. Ngược lại khi khẩu độ lớn, một số lượng ánh sáng các đối tượng xung quanh cũng đi qua lỗ và cùng chiếu tới một điểm trên màn chắn dẫn đến ảnh thu được sẽ bị mờ và mất nét. Trong đó ảnh thu được trên màn chắn sẽ có chiều ngược lại với đối tượng trong không gian 3D.



Hình 15: Mô hình pinhole

Về mặt toán học, mô hình pinhole camera là mô hình toán học mô tả mối quan hệ của các điểm chiếu trong không gian 3D đến mặt phẳng ảnh image plane. Các thông số của mô hình sẽ được miêu tả trong Hình 16. Trong đó Z là khoảng cách từ camera đến đối tượng, f là chiều dài tiêu cự của máy ảnh, được tính từ màn chắn đến bìa mặt hộp chứa lỗ.



Hình 16 Mô hình Pinhole

Từ đó, chúng ta có thể suy ra kích thước của đối tượng dựa vào tỉ lệ tam giác đồng dạng được thể hiện trong phương trình dưới đây:

$$-x = f \frac{X}{Z} \quad (2.1)$$

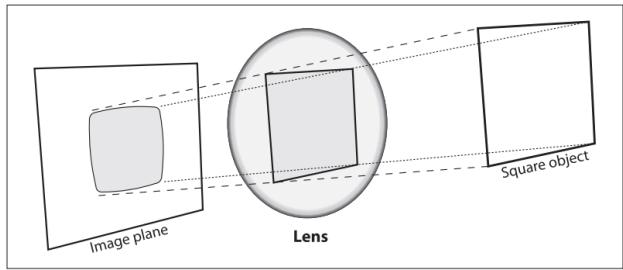
Dấu (-) thể hiện chiều của đối tượng trên màn chẵn.

2.1.2. Lens distortion

Trong lý thuyết, mô hình pinhole camera là một mô hình lý tưởng nên ma trận camera không tính đến sự biến dạng của ống kính. Trong thực tế, ống kính nó sẽ không hoàn hảo. Lý do chính là do nhà sản xuất, ống kính mặt cầu dễ sản xuất hơn so với ống kính có dạng parabol về mặt toán học. Chúng ta cũng khó căn chỉnh chính xác ống kính và hình ảnh một cách cơ học. Để thể hiện chính xác một máy ảnh thực, một mô hình camera sẽ bao gồm hai sự méo dạng chính là méo dạng bán kính (distortion radial) và méo dạng tuyén tính (tangential distortion). Radial distortion sinh ra là kết quả về hình dạng của ống kính, nhưng trái lại tangential distortion sinh ra từ quá trình lắp ráp của toàn bộ camera.

Chúng ta bắt đầu với radial distortion. Ống kính của camera thật thường làm sai lệch đáng kể vị trí của các pixel gần các cạnh của hình ảnh. Trong Hình 17 cho ta thấy một cách trực quan tại sao radial distortion xảy ra. Trên thực tế, một số ống kính rẻ tiền diễn hình thì sẽ gây ra sự méo dạng mạnh hơn.

Đối với sự méo dạng bán kính, sự méo dạng là 0 tại tâm của hình ảnh và tăng khi chúng ta di chuyển về phía đối tượng. Tổng quát, vị trí bán kính của một điểm trên hình ảnh sẽ được dời lại theo phương trình dưới đây:



Hình 17 Radial distortion

$$\begin{aligned}x_{\text{corrected}} &= x(1 + k_1 r^2 + k_2 r^4 + k_3 r^6) \\y_{\text{corrected}} &= y(1 + k_1 r^2 + k_2 r^4 + k_3 r^6)\end{aligned}\quad (2.2)$$

Trong đó:

(x, y) là vị trí ban đầu của điểm bị méo

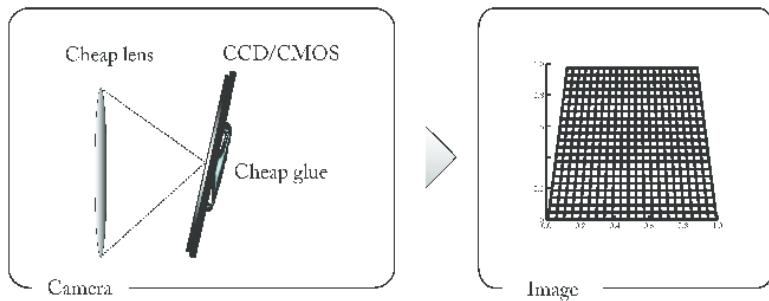
$(x_{\text{corrected}}, y_{\text{corrected}})$ là vị trí mới đã được hiệu chỉnh, không còn bị méo dạng.

k_1, k_2 và k_3 là hệ số méo dạng bán kính của ống kính.

$$r^2 = x^2 + y^2$$

Thông thường, hai hệ số này đã đủ cho việc hiệu chỉnh trong trường hợp của chúng tôi. Đối với một vài sự méo dạng, cũng như trong các ống kính góc rộng, chúng ta có thể chọn ba hệ số bao gồm cả k_3 .

Sự méo dạng phô biến thứ hai là tangential distortion. Sự méo dạng này thì do trong quá trình lắp ráp sản phẩm làm cho các ống kính không song song với mặt phẳng ảnh một cách chính xác, như Hình 18.



Hình 18 Sự méo dạng tiếp tuyến

Biến dạng tuyến tính được đặc trưng bởi hai thông số p_1 và p_2 , như là:

$$\begin{aligned}x_{\text{corrected}} &= x + [2p_1 y + p_2(r^2 + 2x^2)] \\y_{\text{corrected}} &= y + [p_1(r^2 + 2y^2) + 2p_2 x]\end{aligned}\quad (2.3)$$

Như đã giải thích ở trên, trong trường hợp này, chúng tôi chỉ cần tìm bốn hệ số méo dạng. Và tất cả 4 thông số thì hầu hết các quy trình của OpenCV đều sử dụng chúng, chúng thường được thể hiện ở dưới dạng vector. Đó là ma trận 4 hàng 1 cột chứa các thông số k_1, k_2, p_1, p_2 .

$$Distortion_coefficients = [k_1 \quad k_2 \quad p_1 \quad p_2]$$

2.1.3. Homogeneous Coordinates

Trong không gian Euclidean, hai đường thẳng song song giống nhau cùng nằm trên một mặt phẳng thì không thể giao nhau hoặc không thể gặp nhau mãi mãi. Đó là điều hiển nhiên mà mọi người đều biết. Tuy nhiên, nó thì không đúng trong bất kì không gian phản xạ ảnh nào. Ví dụ, đường ray tàu lửa ở Hình 19 trở nên hẹp đi khi nó di chuyển ra xa tầm nhìn của con người. Cuối cùng hai đường ray song song gặp nhau ở chân trời, là một điểm ở vô cực.



Hình 19

Không gian Euclidean (hoặc không gian Descartes) mô tả hình học 2D/3D của chúng ta rất tốt, nhưng chúng không đủ để xử lý không gian phản xạ ảnh (projective space). Thực ra hình học Euclidean là một tập con của hình học phản xạ ảnh. Tọa độ Descartes của một điểm 2D có thể được biểu diễn dưới dạng (x, y) . Nếu điểm này đi xa đến vô cùng thì sao? Điểm ở vô cực sẽ là (∞, ∞) , và nó trở nên vô nghĩa trong không gian Euclidean. Các đường thẳng song song gặp nhau ở vô cùng trong không gian phản xạ ảnh, nhưng không gian Euclidean. Và các nhà toán học đã khám phá ra cách giải quyết vấn đề này là đồng nhất hệ tọa độ (Homogeneous Coordinates).

Homogeneous Coordinates được giới thiệu bởi August Ferdinand Möbius, cho phép thực hiện các phép tính đồ họa và hình học trong không gian phản xạ ảnh. Hệ tọa độ đồng nhất là một cách biểu diễn tọa độ N chiều với N+1 số.

Để tạo tọa độ đồng nhất 2D, đơn giản chúng tôi chỉ cần thêm một biến là w vào tọa độ hiện có. Do đó một điểm trong hệ tọa độ Descarter (x, y) trở thành (x, y, w) trong hệ tọa độ đồng nhất. Và x, y trong tọa độ Descartes được biểu diễn với x, y và w trong tọa độ Hormogeneous như:

$$X = \frac{x}{w}; Y = \frac{y}{w} \quad (2.4)$$

Ví dụ, một điểm trong Dercartes $(1,2)$ trở thành $(1,2,1)$ trong tọa độ đồng nhất. Nếu một điểm $(1,2)$ di chuyển về phía vô cùng, nó sẽ trở thành (∞, ∞) trong hệ tọa độ Descartes. Và nó trở thành $(1,2,0)$ trong tọa độ đồng nhất, vì $(1/0, 2/0) \approx (\infty, \infty)$.

Như đã đề cập trước đây, để mà chuyển từ tọa độ đồng nhất (x, y, w) đến tọa độ Cartesian, chúng ta đơn giản bằng cách chia x và y cho w .

$$(x, y, w) \Leftrightarrow \left(\frac{x}{w}, \frac{y}{w} \right) \quad (2.5)$$

Homogous \Leftrightarrow Cartesian

Khi chuyển hệ tọa độ Homogeneous thành tọa độ Cartesian, chúng ta có thể tìm được một điều quan trọng. Theo dõi ví dụ sau đây:

$$\begin{aligned} (1, 2, 3) &\Rightarrow \left(\frac{1}{3}, \frac{2}{3} \right) \\ (2, 4, 6) &\Rightarrow \left(\frac{2}{6}, \frac{4}{6} \right) \\ (3, 6, 9) &\Rightarrow \left(\frac{3}{9}, \frac{6}{9} \right) \\ &\vdots \\ (1a, 2a, 3a) &\Rightarrow \left(\frac{1a}{3a}, \frac{2a}{3a} \right) = \left(\frac{1}{3}, \frac{2}{3} \right) \end{aligned}$$

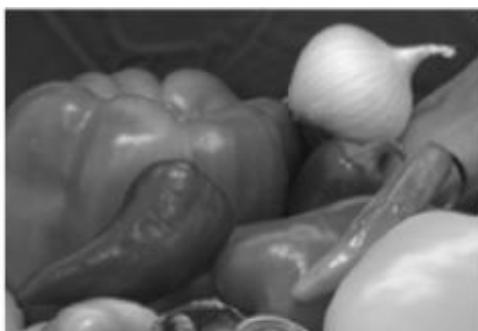
Như bạn có thể thấy, các điểm $(1, 2, 3), (2, 4, 6)$ và $(3, 6, 9)$ tương ứng với cùng một điểm Euclide $(1/3, 2/3)$. Và bất kỳ tích vô hướng nào, $(1a, 2a, 3a)$ thì đều cùng điểm với $(1/3, 2/3)$ trong không gian Euclide. Do đó, những điểm này là đồng nhất vì chúng biểu diễn trong không gian Euclide. Hay nói cách khác tọa độ đồng nhất là bất biến tỉ lệ.

2.2. Cân bằng sáng (histogram equalization)

Trước khi tìm hiểu về cân bằng histogram, chúng ta cần phải biết histogram là gì. Histogram là một biểu đồ tần suất hay đồ thị biểu diễn sự phân bố cường độ sáng của một ảnh. Nó định lượng số lượng pixel cho mỗi giá trị cường độ được xem xét.

Cân bằng histogram là một kỹ thuật được sử dụng để điều chỉnh cường độ của ảnh xám để nâng cao độ tương phản của ảnh.

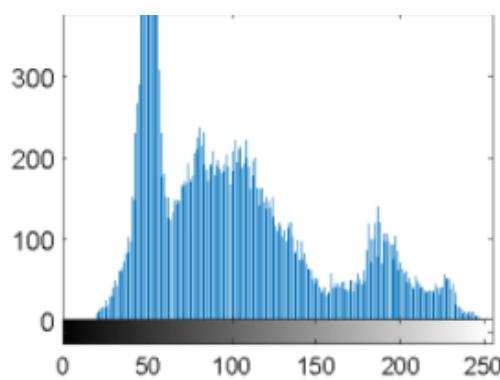
Để làm rõ hơn, từ Hình 22 Đồ thị ảnh chưa cân bằng sáng, bạn có thể thấy rằng các pixel tập trung đa số tại vị trí giữa. Những gì mà cân bằng histogram làm ra mở rộng vùng pixel này. Hãy xem hình bên dưới, vùng ở hai biên của biểu đồ là nơi có cường độ pixel thấp. Sau khi áp dụng cân bằng histogram, chúng tôi thu được một biểu đồ như Hình 23 Đồ thị ảnh đã cân bằng sáng. Dựa vào biểu đồ chúng ta có thể thấy rằng giá trị pixel đã được kéo dãn ra không bị co cụm một khoảng hẹp như trước. Trong thực tế, camera thường chịu ảnh hưởng rất nhiều từ ánh sáng bên ngoài. Chính sự tác động từ ánh sáng làm cho hình ảnh thu được thường bị quá tối hoặc quá sáng. Cân bằng histogram là một thuật toán tiền xử lý rất mạnh mẽ. Phương pháp này cho ta chất lượng ảnh cao, giảm đi sự sai lệch của các dữ liệu. Ta có thể quan sát hình ảnh thu được từ quá trình cân bằng này.



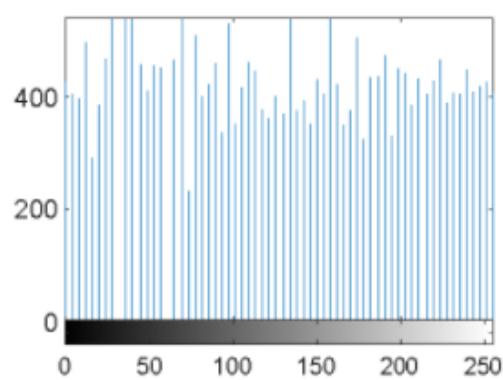
Hình 20 Ảnh chưa cân bằng



Hình 21 Ảnh đã cân bằng sáng



Hình 22 Đồ thị ảnh chưa cân bằng sáng

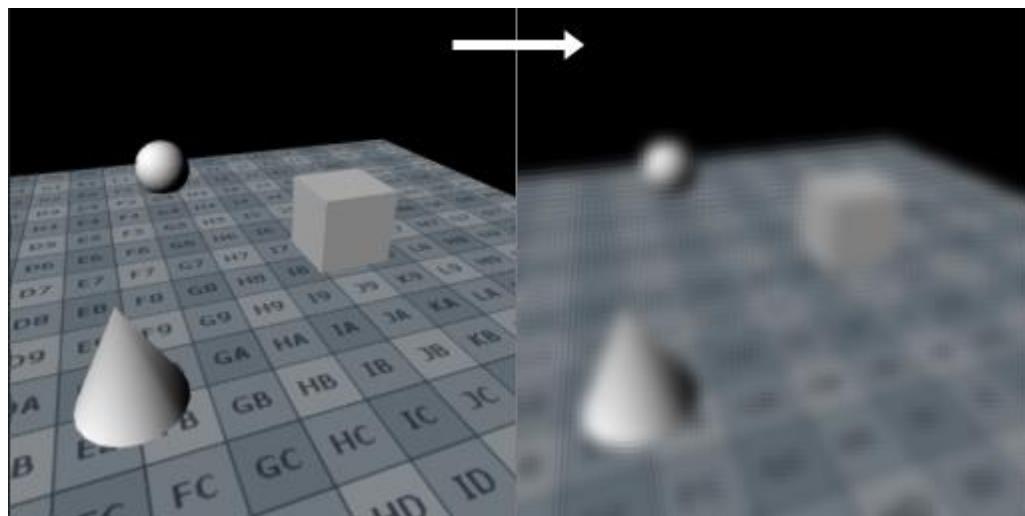


Hình 23 Đồ thị ảnh đã cân bằng sáng

2.3. Gaussian blur

Bộ lọc gaussian [9] là tập hợp các bộ lọc tuyến tính với trọng số được chọn bởi hàm gaussian. Bộ lọc mịn Gaussian là bộ lọc tốt nhất để loại bỏ nhiễu từ phân bố thông thường.

Nó được sử dụng rộng rãi trong các phần mềm đồ họa, thường thì nó được dùng để giảm nhiễu ảnh và làm mờ ảnh. Hiệu ứng hình ảnh của kỹ thuật làm mờ này là một hiệu ứng làm nhèo, làm mịn ảnh giống như khi xem hình ảnh qua màn hình mờ, khác hẳn với hiệu ứng bokeh được tạo ra bởi thấu kính ngoài tiêu cự hoặc bóng của một vật thể dưới ánh sáng thông thường. Làm mượt Gaussian cũng được sử dụng như một giai đoạn xử lý trước trong các thuật toán xử lý ảnh để nâng cao cấu trúc hình ảnh ở các tỷ lệ khác nhau. Ta có thể quan sát ví dụ dưới đây để thấy được sự khác biệt của hình ảnh trước và sau khi áp dụng bộ lọc.



Hình 24 Ảnh trước và sau khi dùng bộ lọc Gaussian blur

Về mặt toán học, áp dụng bộ lọc mờ Gaussian cho một hình ảnh cũng giống như biến đổi hình ảnh với một hàm Gauss. Gaussian có tác dụng làm giảm các thành phần có tần số cao của hình ảnh. Do đó, bộ lọc này cũng có thể được xem như một bộ lọc thông thấp.

Làm mờ Gaussian là một loại bộ lọc làm mờ hình ảnh sử dụng hàm Gauss để tính toán sự chuyển đổi áp dụng cho mỗi pixel trong hình ảnh. Công thức của một hàm pixel trong một chiều là:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}} \quad (2.6)$$

Trong hai chiều, nó tạo ra hai hàm Gaussian, một trong mỗi chiều:

$$G(x, y) = \frac{1}{2\pi\sigma^2} e^{-\frac{x^2+y^2}{2\sigma^2}} \quad (2.7)$$

Trong đó x là khoảng cách từ điểm gốc theo trục hoành, y là khoảng cách từ gốc theo trục tung và σ là độ lệch chuẩn của phân bố Gaussian. Khi được áp dụng theo hai chiều, công thức này tạo ra một bề mặt có đường viền là các đường trong đồng tâm với phân bố Gaussian từ điểm trung tâm.

Các giá trị từ phân phối này được sử dụng để xây dựng ma trận tích chập áp dụng cho ảnh ban đầu. Giá trị mới của mỗi pixel được đặt thành giá trị trung bình có trọng số của vùng lân cận pixel đó. Giá trị của pixel gốc nhận được trọng số cao nhất (có giá trị Gaussian cao nhất) và các pixel lân cận nhận được trọng số nhỏ hơn khi khoảng cách của chúng đến pixel gốc tăng lên. Điều này dẫn đến hiệu ứng làm mờ bao toàn ranh giới và các cạnh tốt hơn các bộ lọc làm mờ khác, đồng nhất hơn.

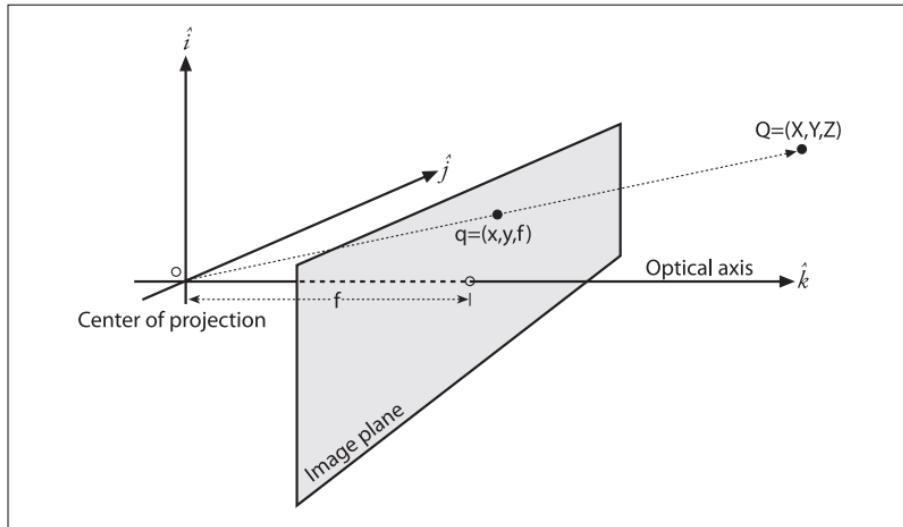
Việc làm mờ ảnh sử dụng mặt nạ Gaussian được sử dụng để định nghĩa không gian tỷ lệ hình ảnh dùng cho các phép nội suy, tính toán các điểm đặc trưng và trong nhiều ứng dụng khác.

2.4. Hiệu chỉnh máy ảnh (camera calibration)

Hiệu chỉnh máy ảnh có nghĩa là xác định các thông số nội của camera. Có thể mở rộng mô hình này để tính toán luôn các thông số méo dạng cầu và các sai số khác nếu ứng dụng yêu cầu độ chính xác cao. Tuy nhiên đối với hầu hết ứng dụng, mô hình đơn giản như pinhole camera là đủ tốt để thực hiện.

Trong dự án này, chúng tôi sử dụng mô hình pinhole camera để thực hiện việc hiệu chỉnh camera. Trong đó chúng tôi sẽ sắp xếp lại mô hình Pinhole camera thành một dạng tương đương. Trong Hình 25, chúng tôi tráo đổi hai mặt phẳng pinhole và mặt phẳng ảnh. Sự khác nhau chính ở đây là ảnh của đối tượng sẽ cùng chiều với đối tượng ngoài thực. Điểm trong pinhole thì được diễn giải lại như là tâm của phép chiếu. Theo cách nhìn này, một điểm nằm trên tia chiếu rời khỏi vật thể và hướng tới tâm của hình chiếu. Điểm tại vị trí giao nhau của mặt phẳng ảnh và trực quang học được gọi là principal point. Trên mặt phẳng ảnh mới này, nó tương đương với image plane được trình bày trong Hình 16, ảnh của đối tượng ở trên mặt phẳng ảnh trong hai mô hình ở hình 1 và hình 2 này thì có kích thước tương đương nhau.

Hình ảnh được tạo ra bằng cách giao các tia chiếu với mặt phẳng ảnh, điều này xảy ra cách tâm chiếu một khoảng cách f . Điều này làm cho mối quan hệ tam giác đồng dạng $x/f = X/Z$ trở nên rõ ràng trực quan hơn trước. Dấu âm biến mất vì ảnh của vật không còn bị hướng xuống.



Hình 25 Điểm $Q = (X, Y, Z)$ chiếu trên mặt phẳng ảnh

Bạn có thể nghĩ rằng điểm principle point tương đương với tâm của hình ảnh, nhưng điều này ngụ ý rằng một số máy ảnh được gắn với tweezers và tube có thể gắn hình ảnh trong máy ảnh của bạn có độ chính xác đến micromet. Trong thực tế tâm của chip thường không nằm trên trục quang học. Do đó chúng tôi giới thiệu 2 thông số mới, c_x và c_y , để mô hình có thể thay thế tọa độ tâm ở trên hình chiếu. Kết quả là một mô đơn giản tương quan với một điểm Q trong không gian vật lý, toàn bộ tọa độ (X, Y, Z) , được chiếu lên màn hình tại một số vị trí pixel được đưa ra bởi $(x_{\text{screen}}, y_{\text{screen}})$ theo các phương trình sau:

$$x_{\text{screen}} = f_x \frac{X}{Z} + c_x, y_{\text{screen}} = f_y \frac{X}{Z} + c_y \quad (2.8)$$

Lưu ý rằng chúng tôi đã giới thiệu hai chiều dài tiêu cự khác nhau, lý do chính là vì đây là các pixels cá nhân có chi phí thấp thường là hình chữ nhật hơn là hình vuông. Tiêu cự f_x là tiêu cự vật lý của ống kính của sản phẩm thực tế và kích thước s_x của các yếu tố riêng lẻ của trình hình ảnh (điều này là có lý bởi vì s_x có đơn vị là pixels/mm trong khi F có đơn vị là mm, điều này có nghĩa rằng f_x trong yêu cầu này có đơn vị là pixels). Đương nhiên, f_y và s_y tương tự. Nó là quan trọng cần ghi nhớ, mặc dù s_x và

s_y không thể đo trực tiếp thông qua bất kì quá trình hiệu chỉnh camera nào, và tiêu cự vật lý F cũng không thể đo trực tiếp. Các sự kết hợp $f_x = Fs_x$ và $f_y = Fs_y$ có thể được tạo ra mà không cần phải tháo máy ảnh và đo trực tiếp các thành phần của nó.

2.5. Hiệu chỉnh stereo camera

Bây giờ chúng ta có tất cả các thông số mà chúng ta cần để hiệu chỉnh từng camera, nhưng chúng ta vẫn cần để hiệu chỉnh các camera cùng nhau. Bằng cách tập trung vào tất cả các thông số đã được tính toán trước đó (các điểm 2D, 3D của chessboard cho cả hai camera, ma trận camera cho cả hai camera, hệ số méo cho cả hai camera), chúng tôi có thể thu được ma trận xoay và vector tịnh tiến T giữa hệ tọa độ của camera thứ nhất và thứ hai:

$$\begin{aligned} R_2 &= RR_1 \\ T_2 &= RT_1 + T \end{aligned} \quad (2.9)$$

Trong đó R_i và T_i hướng và vị trí của camera i , R và T là ma trận tính toán mối quan hệ về hướng và vị trí của một camera với camera khác.

Sau đó, các bản đồ hiệu chỉnh sự biến đổi được tính toán từ các thông số đó. Cùng với ma trận xoay, các thông số này sẽ được sử dụng để đưa mặt phẳng ảnh của 2 camera về cùng một mặt phẳng.

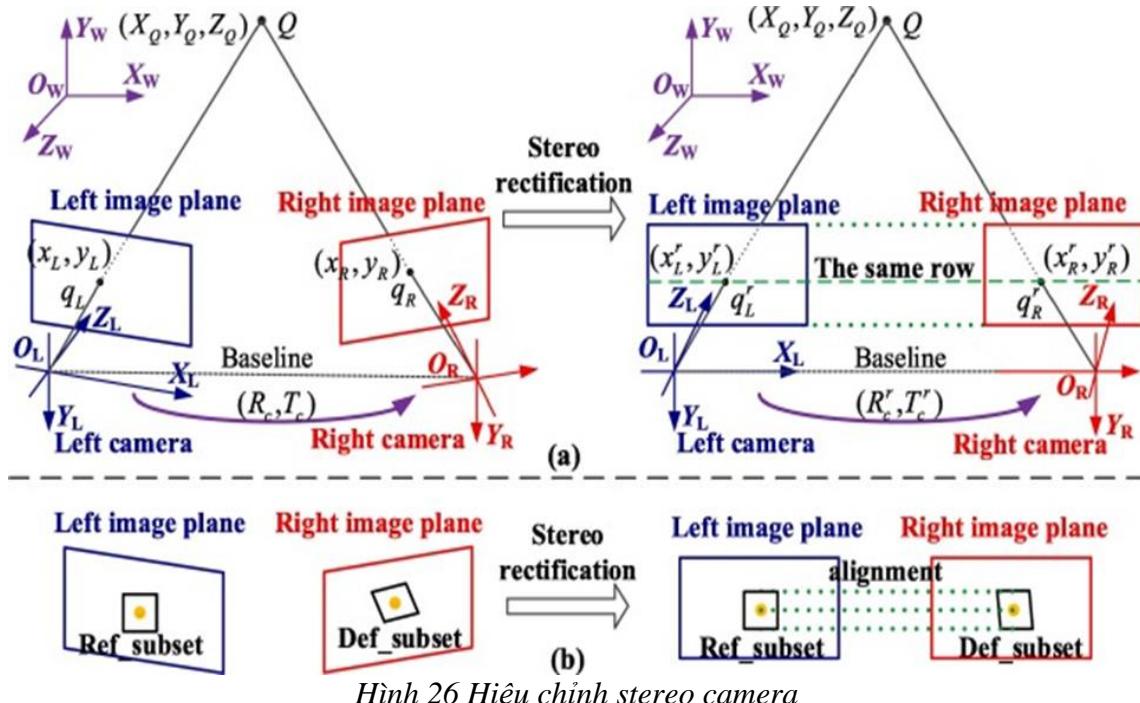
Bước này thì cần thiết, bởi vì thuật toán đối sánh stereo (Stereo Matching algorithms) được sử dụng ở phần sau, nên chúng ta cần tất cả các đối tượng tương ứng mà nằm trong góc nhìn của 2 camera, phải nằm trên cùng một đường pixels nằm ngang. Trong thực tế, xem xét rằng mặt phẳng epipolar giao với một mặt phẳng ảnh tạo thành một đường duy nhất đối với cả hai camera, việc hiệu chỉnh cho phép đường epipolar song song, đơn giản hóa vấn đề tương đương của stereo, vì thuật toán của chúng tôi sẽ tìm kiếm theo phương ngang cho các nhóm pixel tương ứng với cùng một đối tượng trong hai hình ảnh, kết quả thu được bản đồ chênh lệch(disparity maps).

Các hàm chính của thư viện OpenCV được sử dụng trong quá trình này là:

‘cv2.stereoCalibrate’: hàm này yêu cầu ma trận camera và hệ số méo đối với hai camera. Ngõ ra là vectors xoay R và T , hai thông số này thể hiện mối quan hệ giữa hai hệ tọa độ của camera.

‘cv2.stereoRectify’: sử dụng R và T, điều đó cho phép tính toán ma trận xoay đổi với 2 camera. Một khía cạnh chúng được áp dụng, cả hai mặt phẳng ảnh sẽ song song với nhau.

Sự ảnh hưởng của việc hiệu chỉnh ảnh được thể hiện trong Hình 26: các mặt phẳng ảnh đã được xoay để các chi tiết được hiệu chỉnh và ta có nhìn thấy dọc theo các đường epipolar giống nhau trên cả hai hình ảnh trái và phải.

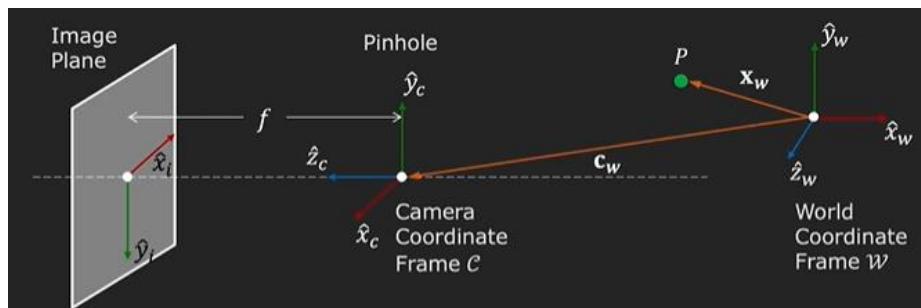


Hình 26 Hiệu chỉnh stereo camera

2.6. Geometry of image formation

Cho một điểm 3D là P trong không gian, chúng tôi muốn tìm tọa độ pixel (u, v) của điểm 3D này trong hình được chụp bởi một máy ảnh.

Có 3 hệ tọa độ được thiết lập trong hệ này, nó được thể hiện ở hình dưới đây:



Hình 27 Toạ độ trong không gian

Để xác định vị trí của các điểm đối tượng trong không gian chúng ta cần định nghĩa hệ tọa độ cho không gian. Chúng ta có một hệ tọa độ không gian w, một hệ tọa độ camera là c và một điểm nằm trong cả hai hệ tọa độ không gian và camera là p. Dựa vào Hình

27 chúng ta có thể thấy rằng trục z của hệ tọa độ camera nằm trùng với trục quang học(optical axis) của camera. Và ta gọi F là tiêu cự của camera, là khoảng cách từ tâm của hình chiếu đến mặt phẳng ảnh(image plane).

Bây giờ nếu chúng ta biết mối quan hệ về vị trí và hướng của hệ tọa độ camera đến hệ tọa độ không gian thì chúng ta có thể viết một biểu thức đưa điểm p trong hệ tọa độ không gian đến điểm ảnh xi nằm trên image plane. Chúng ta bắt đầu với một điểm trong tọa độ không gian $X_w = [x_w \ y_w \ z_w]^T$, sau đó chúng ta sẽ biến đổi Xw trong tọa độ tham chiếu không gian thành $X_c = [x_c \ y_c \ z_c]^T$ trong tọa độ tham chiếu camera. Hãy lưu ý rằng đây là một sự biến đổi từ điểm Xw trong không gian 3D đến 3D. Một khi chúng ta có Xc, chúng ta có thể áp dụng phép chiếu phối cảnh để chiếu điểm Xw lên tọa độ ảnh(image coordinates), đó là tọa độ 2D, Xi, ở trên mặt phẳng ảnh. Bởi vì toàn bộ mô hình này đi từ tọa độ không gian đến tọa độ ảnh, 3D đến 2D được gọi mô hình hình ảnh chuyển tiếp. Và chúng tôi sẽ sử dụng nó để phát triển một mô hình cho camera hoàn toàn tuyến tính. Vì vậy chúng tôi sẽ bắt đầu với phần sau của hai phép biến đổi này, đó là phép chiếu phối cảnh (perspective projection).

Chúng ta đã biết một phương trình perspective projection đã được sử dụng rộng rãi cho đến nay là:

$$\frac{x_i}{f} = \frac{x_c}{z_c} \text{ và } \frac{y_i}{f} = \frac{y_c}{z_c} \quad (2.10)$$

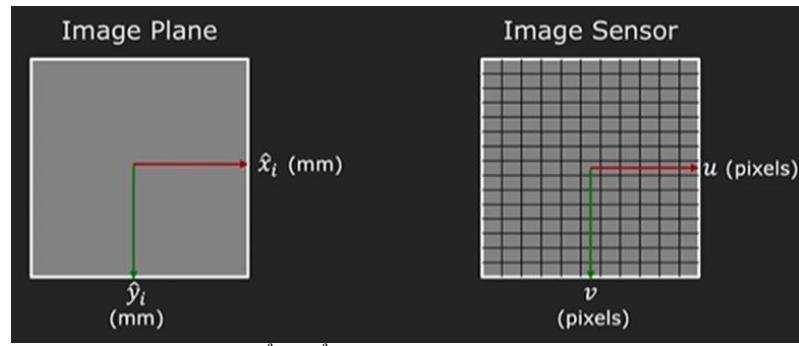
Chúng ta có thể viết lại phương trình 1.8 để thu được x_i và y_i như sau:

$$x_i = f \frac{x_c}{z_c} \text{ và } y_i = f \frac{y_c}{z_c} \quad (2.11)$$

Trong đó:

x_i, y_i là tọa độ hình chiếu của điểm p lên image plane

Tiếp theo sẽ có một vấn đề chúng ta cần xem xét trên image plane, giả sử chúng ta biết tọa độ điểm ảnh của điểm trên mặt phẳng ảnh 2D. Tuy nhiên đơn vị của nó là mm, giống với đơn vị của điểm p xác định trong hệ tọa độ camera. Nhưng trong thực tế, những gì chúng ta có là một cảm biến ảnh, được sử dụng để chụp ảnh. Và cảm biến ảnh này chúng ta biết chúng có đơn vị là pixels.

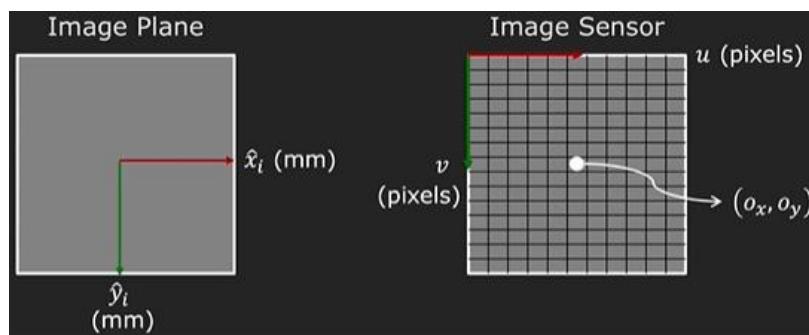


Hình 28 Chuyển đổi từ kích thước sang tọa độ pixel

Gọi u, v là hệ tọa độ của cảm biến máy ảnh ở đơn vị pixels, m_x, m_y lần lượt là cường độ pixels/mm theo các chiều x và y , khi đó tọa độ pixels sẽ là:

$$u = m_x x_i = m_x f \frac{x_c}{z_c}, v = m_y y_i = m_y f \frac{y_c}{z_c} \quad (2.12)$$

Vì chúng ta đi từ milimeters đến pixels và giá trị m_x, m_y này thì không biết. Chúng là một phần của quá trình hiệu chỉnh. Có một điều nữa mà chúng ta cần quan tâm đó là cho đến nay chúng tôi đã giả định rằng chúng tôi biết vị trí trung tâm của hình ảnh là pixels $(0,0)$ tương ứng với tâm của hình ảnh, nơi giao nhau của trực quang học là mặt phẳng ảnh (image plane). Nhưng không có lí do gì để biết điều đó là cần và để chúng ta biết điểm đó ở đâu, nên ta gọi điểm đó là điểm nguyên tắc (principal point). Trong thực tế, khi chúng ta chụp ảnh, tọa độ tham chiếu của ảnh thường nằm tại vị trí 1 trong 4 góc của hình ảnh, nó có thể là góc trên bên trái, góc trên bên phảiVà vị trí này sẽ có tọa độ pixels là $(0,0)$. Chúng ta thực sự không biết điểm principal ở đâu nên chúng tôi sẽ gọi vị trí của nó là o_x, o_y . Vì vậy bây giờ chúng ta phải hiệu chỉnh phương trình phép chiếu ánh xạ là:



Hình 29 Tọa độ điểm principal

$$u = m_x f \frac{x_c}{z_c} + o_x, v = m_y f \frac{y_c}{z_c} + o_y \quad (2.13)$$

Trong công thức (2.13) , chúng ta có thể kết hợp $m_x f$ bằng f_x và $m_y f$ bằng f_y là tiêu cự của thấu kính theo các chiều x và y trong tọa độ có đơn vị là pixels.

$$u = f_x \frac{x_c}{z_c} + o_x, v = f_y \frac{y_c}{z_c} + o_y \quad (2.14)$$

(f_x, f_y, o_x, o_y) là các thông số nội của camera, chúng đại diện cho hình học bên trong của camera. Một điều cần lưu ý ở đây là chúng ta có một mô hình, nhưng đó là một mô hình phi tuyến tính. Và chúng tôi biết rằng để quá trình tính toán trở đo đạt trơ nêu dễ dàng hơn thì chúng tôi phải chuyển nó về mô hình dạng tuyến tính. Để có thể thực hiện quá trình chuyển đổi này, chúng tôi đã sử dụng hệ tọa độ đồng nhất (Homogenous Coordinates).

Đồng nhất sự đại diện của một điểm 2D $u = (u, v)$ là một điểm 3D $\tilde{u} = (\tilde{u}, \tilde{v}, \tilde{w})$. Ta có tọa độ đồng nhất của (u, v) :

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \equiv \begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} \equiv \begin{bmatrix} z_c u \\ z_c v \\ z_c \end{bmatrix} = \begin{bmatrix} f_x x_c + z_c o_x \\ f_y y_c + z_c o_y \\ z_c \end{bmatrix} = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \quad (2.15)$$

$$\text{Trong đó } (u, v) = \left(\frac{\tilde{u}}{\tilde{w}}, \frac{\tilde{v}}{\tilde{w}} \right)$$

Từ (2.15) ta suy ra ma trận nội và ma trận hiệu chỉnh của camera là:

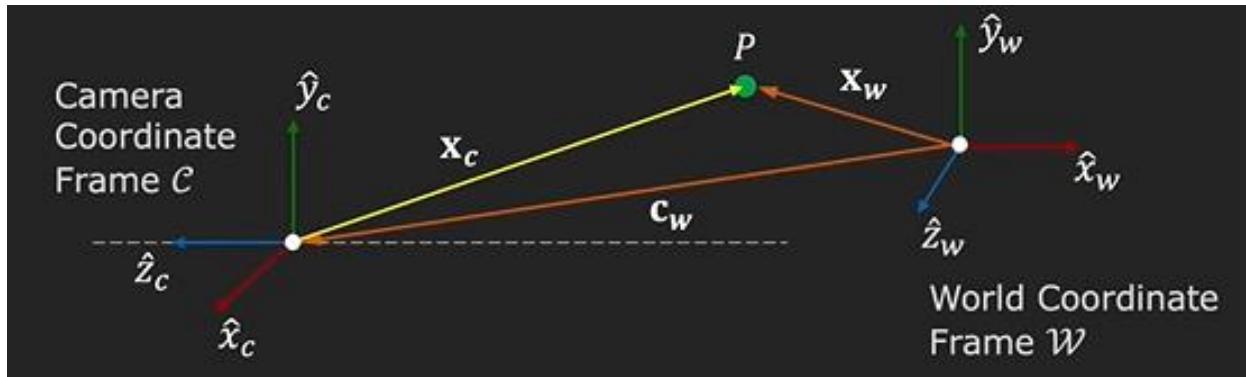
Intrinsic matrix:

$$M_{\text{int}} = [k \mid 0] = \begin{bmatrix} f_x & 0 & o_x & 0 \\ 0 & f_y & o_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \quad (2.16)$$

Calibration matrix:

$$K = \begin{bmatrix} f_x & 0 & o_x \\ 0 & f_y & o_y \\ 0 & 0 & 1 \end{bmatrix} \quad (2.17)$$

Các thông số tiếp theo mà chúng ta phải tìm thông qua quá trình biến đổi từ điểm của đối tượng trong tọa độ không gian 3D đến tọa độ 3D trong camera. Quá trình biến đổi này thể hiện một ma trận bao gồm ma trận xoay R và ma trận dịch chuyển t .



Hình 30 Phép chiếu tọa độ 3D đến tọa độ 3D

Cho các thông số ngoại (R, c_w) của camera, vị trí tâm của điểm Pin tọa độ tham chiếu không gian là:

$$X_c = R(x_w - c_w) = Rx_w - Rc_w = Rx_w + t, (t = -Rc_w)$$

$$X_c = \begin{bmatrix} x_c \\ y_c \\ z_c \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} \\ r_{21} & r_{22} & r_{23} \\ r_{31} & r_{32} & r_{33} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \end{bmatrix} + \begin{bmatrix} t_x \\ t_y \\ t_z \end{bmatrix} \quad (2.18)$$

Ta có thể viết lại ma trận này áp dụng tọa độ đồng nhất:

$$\tilde{X}_c = \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix} \quad (2.19)$$

Ma trận ngoại:

$$M_{ext} = \begin{bmatrix} R_{3x3} & t \\ 0_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.20)$$

Từ (2.16), (2.20) ta thu được mối quan hệ từ phép biến đổi một điểm trong không tọa độ không gian đến điểm ảnh 2D trên tọa độ ảnh.

Camera to pixels:

$$\tilde{u} = M_{int} \tilde{x}_c$$

Không gian to camera:

$$\tilde{X}_c = M_{\text{ext}} \tilde{x}_w$$

Kết hợp hai phương trình ở trên, chúng ta có được ma trận ánh xạ P:

$$\tilde{\mathbf{u}} = \mathbf{M}_{\text{int}} M_{\text{ext}} \tilde{x}_w = P \tilde{x}_w$$

$$\begin{bmatrix} \tilde{u} \\ \tilde{v} \\ \tilde{w} \end{bmatrix} = \begin{bmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

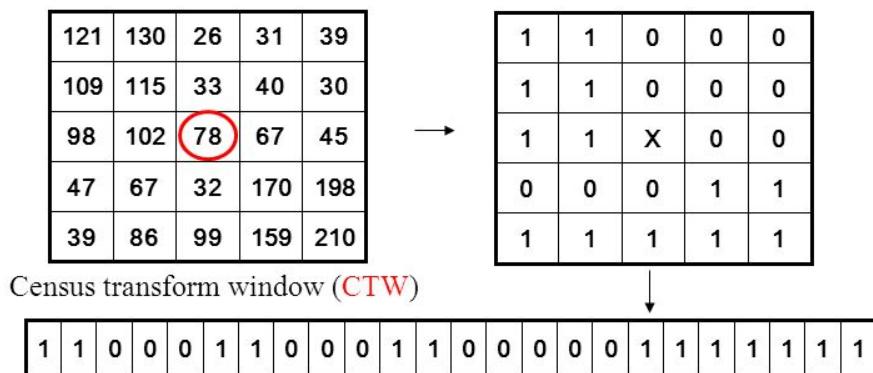
2.7. Stereo matching:

2.7.1. Tóm quan:

Phương pháp stereo matching là quá trình phát hiện những điểm ảnh tương ứng giữa cặp ảnh phù hợp với điểm 3D thực tế từ đó tính toán được bản đồ chênh lệch (disparity maps). Cặp hình ảnh được tinh chỉnh từ việc áp dụng hình học Epipolar sẽ đơn giản hóa quá trình stereo matching vì chỉ cần tìm kiếm các pixel trên đường thẳng epipolar.

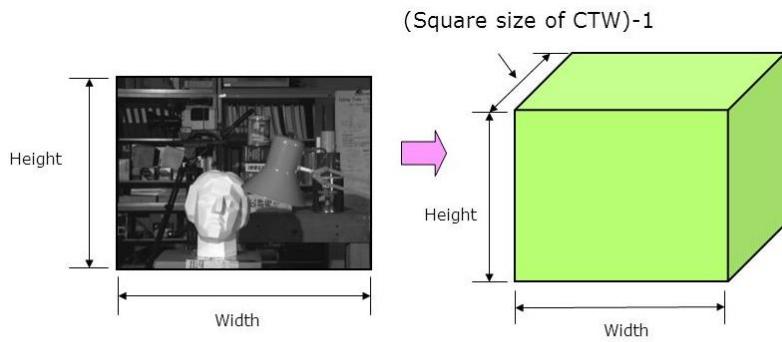
Áp dụng phương pháp Stereo matching để tính toán bản đồ chênh lệch như sau:

Bước 1: Tính toán matching cost [10][11] áp dụng biến đổi census và khoảng cách hamming:



FCV 2006

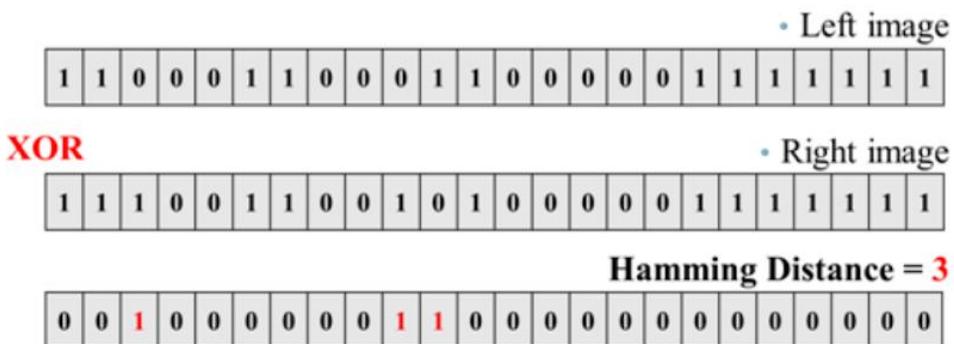
Hình 31 Biến đổi census.



FCV 2006

Hình 32 Dữ liệu sau biến đổi census.

- Census transform [12] chuyển đổi giá trị cường độ pixel thành 0 và 1.
- Sau đó biến đổi thành vector 1 chiều.
- Kết quả phép biến đổi tạo thành dữ liệu với kích thước (image size * vector)



Hình 33 Hamming Distance.

- Hamming distance của 2 vector census đã biến đổi sẽ được dùng để tìm ra vị trí pixel tương ứng.

Bước 2: Tổ hợp các giá trị matching cost (cost aggregation) và tính toán bản đồ chênh lệch dựa theo phương pháp Semi-global matching (trình bày ở mục 2.7.2).

Bước 3: Hậu xử lý cho bản đồ chênh lệch dùng bộ lọc trung vị (trình bày ở mục 2.8).

2.7.2. Phương pháp Semi-global matching:

Semi-global matching (SGM) [13] là một thuật toán thị giác máy tính để ước tính bản đồ chênh lệch từ một cặp ảnh được chụp từ stereo camera và đã được chỉnh sửa. Thuật toán được giới thiệu vào năm 2005 do Heiko Hirschmuller nghiên cứu ra khi ông đang làm việc tại trung tâm hàng không vũ trụ Đức. Với thời gian chạy có thể dự đoán được, sự cân bằng giữa chất lượng kết quả và thời gian tính toán cũng như khả năng phù Sô hiệu: HD/QT-PKHCN-QHQT-NCKHSV/00 Lần soát xét: 00 Ngày hiệu lực: 01/4/2020 Trang: 34/90

hợp để triển khai song song nhanh chóng trong ASIC hoặc FPGA. Nó đã được ứng dụng rộng rãi trong các ứng dụng sử dụng stereo camera theo thời gian thực như robot và trình điều khiển nâng cao hệ thống hỗ trợ.

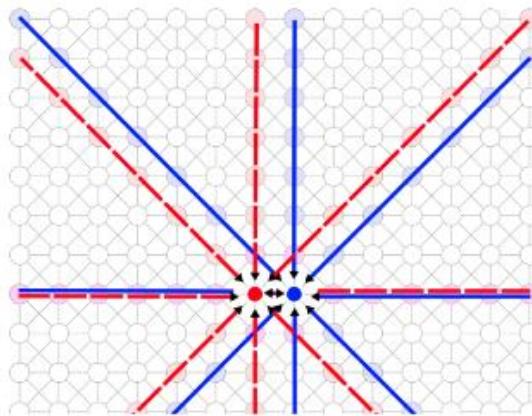
Các bước thực hiện thuật toán semi-global matching [14] là:

Bước 1: Khởi tạo các hướng di chuyển hội tụ về pixel p.

Bước 2: Trong quá trình di chuyển theo các hướng sẽ tạo ra tập hợp các điểm pixel và các Cost (đã tính ở bước 1 phần 2.7.1) của pixel này sẽ đóng góp một phần vào cost của pixel p.

Bước 3: Tổng hợp cost của các hướng đi tại pixel p.

Bước 4: Tìm pixel có giá trị cost thấp nhất và sau đó tạo thành bản đồ chênh lệch (disparity maps).



Hình 34 Phương pháp semi-global matching

Thuật toán SGM chính là việc cực tiểu hóa hàm E cho bản đồ chênh lệch D như sau:

$$E(D) = \sum_p \left(C(p, D_p) + \sum_{q \in N_p} P_1 I[D_p - D_q = 1] + \sum_{q \in N_p} P_2 I[D_p - D_q > 1] \right) \quad (2.21)$$

Với các hướng r thì tổng giá trị cost tại pixel p của disparity thứ D có dạng như sau:

$$S(p, d) = \sum_r L_r(p, d) \quad (2.22)$$

Trong đó:

$$L_r(p,d) = C(p,d) + \min(L_r(p-r,d), L_r(p-r,d-1) + P_1, L_r(p-r,d+1) + P_1, \min_i L_r(p-r,i) + P_2) - \min_k L_r(p-r,k)$$

Với

$C(p,d)$ là giá trị cost hiện tại của pixel p tại disparity thứ d.

Hệ số P_1, P_2 được cài đặt cho sự thay đổi độ chênh lệch disparity. Ví dụ với $d=8$ thì giá trị $d=7$ và $d=9$ thì sẽ được thêm hệ số P_1 , còn đối với các trường hợp nhỏ hơn $d-1$ hoặc lớn hơn $d+1$ thì sẽ được thêm hệ số P_2 , hệ số $P_1 \leq P_2$.

$L_r(p-r,d)$ là giá trị cost của pixel trước đó theo hướng r tại disparity map thứ d.

$L_r(p-r,d-1) + P_1$ là giá trị cost của pixel trước đó theo hướng r tại bản đồ chênh lệch thứ $d-1$, nên thêm hệ số P_1 vào.

$L_r(p-r,d+1) + P_1$ là giá trị cost của pixel trước đó theo hướng r tại bản đồ chênh lệch thứ $d+1$, nên thêm hệ số P_1 vào.

$\min_i L_r(p-r,i) + P_2$ là tìm ra giá trị cost nhỏ nhất của pixel trước đó theo hướng r tại bản đồ chênh lệch nhỏ hơn $d-1$ và lớn hơn $d+1$, nên thêm hệ số P_2 vào.

$\min_k L_r(p-r,k)$ là tìm ra giá trị cost nhỏ nhất của pixel trước đó theo hướng r trên tất cả disparity.

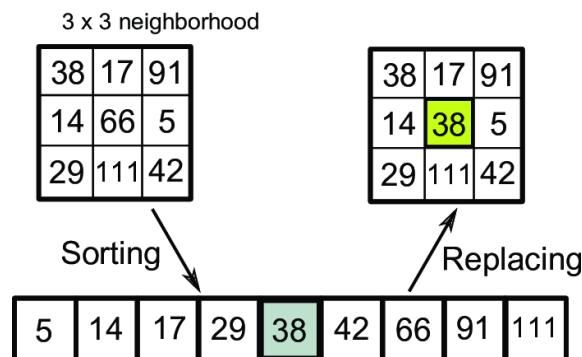
Sau khi tính được tổng cost của tất cả các hướng. Ta sẽ dựa theo cách tiếp cận winner-takes-all làm disparity map cuối cùng.

2.8. Bộ lọc trung vị (Median filter)

Bộ lọc trung vị là một trong những bộ lọc thống kê nổi tiếng do nó có hiệu suất tốt đối với một số loại nhiễu cụ thể như nhiễu muối tiêu (salt-pepper noise), nhiễu đốm (speckle noise). Theo bộ lọc trung vị thì pixel trung tâm của vùng lân cận MxM được thay thế bằng giá trị trung vị của cửa sổ tương ứng. Lưu ý rằng điểm ảnh nhiễu được coi là rất khác với điểm trung vị.

Ý tưởng chính của thuật toán là sử dụng một cửa sổ lọc hay mặt nạ có nxn và lần lượt quét qua từng điểm ảnh của ảnh ban đầu chưa nhiễu. Sau khi áp mặt nạ này vào tại vị trí của mỗi điểm ảnh thì các giá trị pixel trong vùng của mặt nạ sẽ được sắp xếp tăng

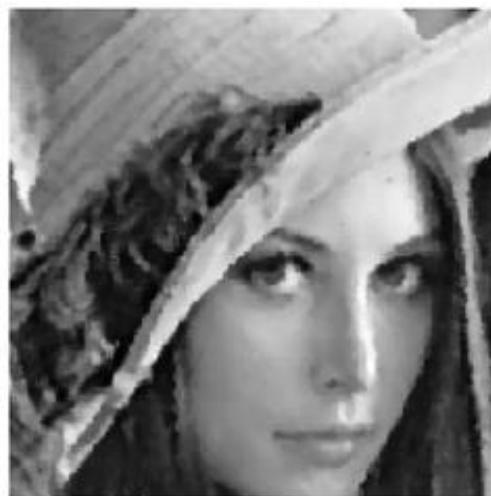
dần hoặc giảm dần về giá trị cường độ. Cuối cùng, gán điểm ảnh nằm ở vị trí chính giữa của các giá trị vừa được sắp xếp, gán cho giá trị điểm ảnh đang xét. Hình 35 mô tả quá trình hoạt động của bộ lọc trung vị.



Hình 35 Ví dụ về hoạt động của bộ lọc trung vị



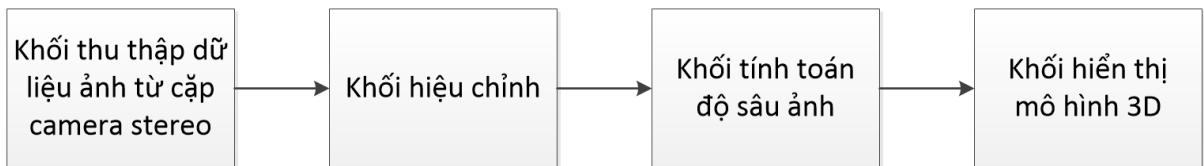
Hình 36 Ảnh gốc



Hình 37 Ảnh đã được lọc

Chương 3: XÂY DỰNG MÔ HÌNH 3D VẬT THỂ

Để xây dựng được mô hình 3D của vật thể, nhóm phải cần nhiều bước và nhiều công đoạn như lấy dữ liệu ảnh, xử lý dữ liệu ảnh, tạo bản đồ độ sâu dựa vào màu sắc, tạo đám mây điểm ảnh và hiển thị chúng. Do đó để cụ thể hơn về mô hình, trong chương này nhóm sẽ tạo ra từng khối với chức năng nhiệm vụ riêng biệt. Ta có sơ đồ khái:



Hình 38 Sơ đồ khái quát quá trình xây dựng mô hình 3D

- Khối thu thập dữ liệu ảnh từ cặp camera stereo:

Khối này có nhiệm vụ lấy dữ liệu ảnh bao gồm 2 ảnh màu từ 2 camera, đồng thời chụp 20 tấm ảnh chessboard cho mỗi camera. Với điều kiện cùng không gian như nhau và sẽ chụp nhiều đối tượng để so sánh kết quả. Ở đây khối này sẽ cho ta kết được ma trận bên trong và bên ngoài của mỗi camera từ đó tạo dữ liệu cho khối hiệu chỉnh.

- Khối hiệu chỉnh:

Điều chỉnh các thông số ma trận bên trong và bên ngoài camera kết hợp so sánh đối chiếu đặc trưng của chúng và dùng các phương pháp xử lý ảnh để điều chỉnh ánh sáng, màu sắc, kích thước để cho hai bức ảnh khớp nhau.

- Khối tính toán độ sâu ảnh

Khối này lấy ý tưởng từ hai mắt của con người, nếu ta che một mắt và nhìn một mắt cho một đối tượng thì khi làm ngược lại thì ta sẽ thấy đối tượng dịch chuyển một đoạn, nếu khoảng cách dịch chuyển này lớn thì vật đó là ở gần chúng ta hơn và nếu khoảng cách dịch chuyển nhỏ thì vật ở xa hơn. Từ đó, ta có thể tính toán được độ sâu của đối tượng so với các vật xung quanh.

- Khối hiển thị mô hình 3D

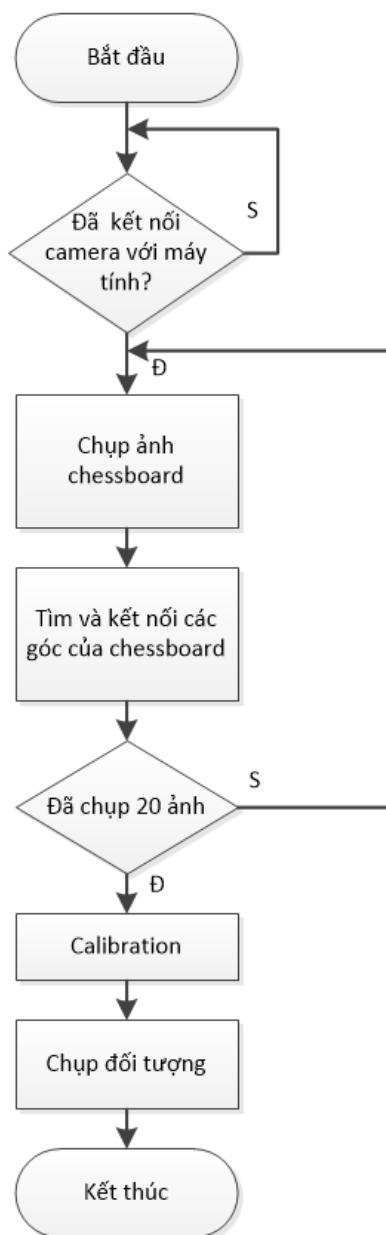
Sau khi tạo ra được bản đồ độ sâu, ta tiến hành xử lý giữa bức ảnh RGB và độ sâu của từng khung hình để tìm kết nối tương đồng 2D giữa 2 ảnh liên tiếp. Mỗi vị trí toạ độ pixel ta sẽ có được giá trị độ sâu tương đương. Nhờ vậy ta thu được đám mây sau

hiệu chỉnh. Khi thu được đám mây điểm ảnh ta tiếng hành hiển thị lên để quan sát, kiểm tra thực nghiệm so với kết quả nghiên cứu.

3.1. Khôi lấy dữ liệu ảnh từ camera

Bao gồm các công đoạn sau:

- Giữ đối tượng cố định, cụ thể và vật thể (hoặc nhiều vật thể), không gian xung quanh
- Sử dụng cặp stereo camera để chụp 20 tấm ảnh chessboard cho mỗi camera với khoảng cách từ camera tới đối tượng được giữ cố định cho việc chụp ảnh đối tượng. Để hiểu chi tiết hơn ta theo dõi lưu đồ thu thập dữ liệu từ stereo camera.



Hình 39 Lưu đồ thu thập dữ liệu từ stereo camera

Đầu tiên ta kết nối 2 đầu kết nối USB vào máy tính, ở đây nhóm dùng máy tính Windows 10 Pro, thiết lập camera cho máy tính. Sau khi thiết lập thành công, camera cho phép chụp ảnh thì ta đặt cặp máy ảnh theo vị trí đã cho trước và cố định khoảng cách của chúng so với đối tượng. Sau đó dùng code điều chỉnh và tiến hành chạy chương trình để thu thập ảnh chessboard rồi tới đối tượng. Sau khi thu thập dữ liệu xong thì qua khôi hiệu chỉnh, để xử lý đầu vào là ma trận của camera.

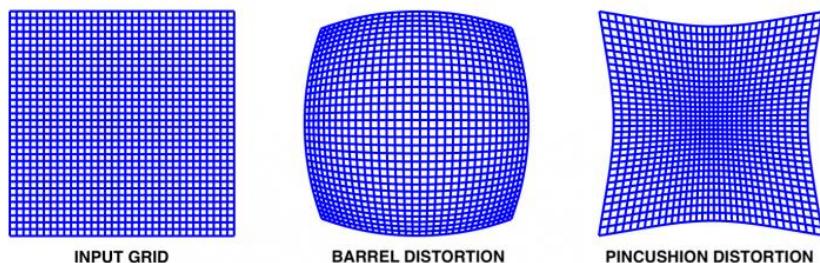
3.2. Khôi hiệu chỉnh

Trước hết, chúng ta cần hiệu chỉnh hai máy ảnh một cách riêng biệt: điều này là do ta dùng ống kính nên sẽ xảy 2 loại biến dạng chính:

- Méo xuyên tâm: Loại biến dạng này thường xảy ra do sự bê cong ánh sáng không đồng đều. Các tia ở gần các cạnh của thấu kính uốn cong hơn các tia ở gần tâm của thấu kính. Do sự biến dạng xuyên tâm, các đường thẳng trong thế giới thực dường như bị cong trong hình ảnh. Tia sáng bị dịch chuyển hướng tâm vào trong hoặc ra ngoài khỏi vị trí lý tưởng của nó trước khi chạm vào cảm biến hình ảnh. Có hai loại hiệu ứng méo xuyên tâm:

- + Hiệu ứng biến dạng Barrel, tương ứng với dịch chuyển xuyên tâm âm
- + Hiệu ứng biến dạng Pincushion, tương ứng với dịch chuyển hướng tâm dương.

- Biến dạng tiếp tuyến: Điều này thường xảy ra khi màn hình hoặc cảm biến hình ảnh ở một góc nghiêng so với ống kính. Do đó, hình ảnh dường như bị nghiêng và kéo dài.



Hình 40 Các loại méo dạng ảnh

Để giải quyết những vấn đề đó, chúng ta cần tìm hệ số méo và cả ma trận camera (chứa tiêu cự và tâm quang học).

Hệ số méo:

$$[k_1 \quad k_2 \quad p_1 \quad p_2] \quad (3.1)$$

Ma trận máy ảnh:

$$\begin{bmatrix} f_x & 0 & cx \\ 0 & f_y & cy \\ 0 & 0 & 1 \end{bmatrix} \quad (3.2)$$

Trong đó:

f_x, f_y : lần lượt là tiêu cự ngang và dọc

cx, cy : lần lượt là toạ độ pixel ngang và dọc của điểm chính giữa

k_1, k_2 : hệ số méo xuyên tâm

p_1, p_2 : hệ số méo tiếp tuyến

Bây giờ chúng ta có tất cả các thông số mà chúng ta cần để hiệu chỉnh các camera riêng lẻ, nhưng chúng ta vẫn cần phải hiệu chỉnh stereo camera. Bằng cách thu thập tất cả các thông số đã tính toán trước đó (ma trận cho cả hai camera, hệ số biến dạng cho cả hai camera), nhóm có thể thu được ma trận xoay R và vector tịnh tiến T giữa máy ảnh thứ nhất và thứ hai trong hệ thống tọa độ:

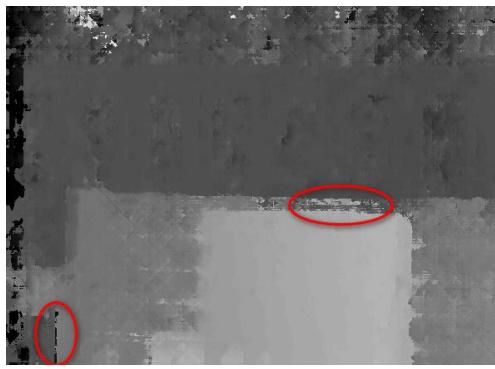
$$\begin{aligned} R_2 &= R.R_1 \\ T_2 &= R.T_1 + T \end{aligned} \quad (3.3)$$

Sau khi đã điều chỉnh 2 ảnh trái và phải, tuy nhiên còn gặp nhiều vấn đề như ánh sáng 2 ảnh khác nhau, vật thể trong suốt, ... ta sử dụng cân bằng histogram, sau đó làm mờ ảnh bởi Gaussian kernel để thu được kết quả 2 ảnh tương đương nhau để chuẩn bị tốt cho khâu tính toán độ sâu.

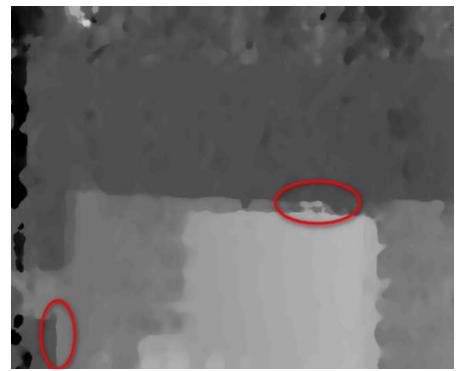
3.3. Khối tính toán độ sâu ảnh

Như ở chương 2 đã trình bày, việc tính toán disparity dựa vào thuật toán Semi global matching. Sau khi sử dụng “matching cost” ta thu được các bản đồ độ sâu. Các phương pháp tính matching cost có thể dựa vào SAD, SSD, NCC. Trong chương trình nhóm sử dụng census transform và hamming distance.

Sau khi thu được disparity thì xảy ra vấn đề như là các giá trị bị nhiễu và hiếu nhầm là các giá trị disparity lỗi nên nhóm sử dụng thêm phương pháp lọc trung vị để giải quyết vấn đề đó.



Hình 41 Disparity trước khi lọc trung vị



Hình 42 Disparity sau khi lọc trung vị

3.4. Khôi hiển thị mô hình 3D

Bản đồ độ sâu là bước cuối cùng để tạo ra mô hình 3D mong muốn và nó chỉ đơn giản thể hiện sự liên kết của phép đo độ sâu thực (mét) với giá trị chênh lệch của mỗi pixel.

Đầu tiên ta cần tính ma trận phản chiếu chênh lệch độ sâu (Q):

$$Q = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 0 & f \\ 0 & 0 & \frac{1}{T_x} & \frac{c_x - c'_x}{T_x} \end{bmatrix} \quad (3.4)$$

Với: cx là tọa độ pixel ngang của điểm chính giữa của một trong 2 máy ảnh.

cy là tọa độ pixel dọc của điểm chính giữa của cùng một máy ảnh.

c'x là tọa độ pixel ngang của điểm chính của máy ảnh còn lại

Tx là độ dài đường cơ sở, tương ứng với khoảng cách giữa 2 máy ảnh.

f là khoảng cách giữa tiêu điểm của máy ảnh và mặt phẳng ảnh của nó.

Và giá trị phép chiếu được tính như sau:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = Q \cdot \begin{bmatrix} u \\ v \\ disparity(u, v) \\ 1 \end{bmatrix} \rightarrow 3Dpoint = \begin{bmatrix} \frac{X}{Y} \\ \frac{Y}{W} \\ \frac{Z}{W} \end{bmatrix} \quad (3.5)$$

Với: u và v là tọa độ của pixel trong ảnh.

Disparity (u, v) là giá trị chênh lệch liên quan đến pixel ở vị trí u, v.

X, Y, Z và W là các tọa độ không gian 3D, với W là được đề cập đến tỷ lệ

Mỗi pixel đã được dịch thành một tọa độ không gian 3D thực tính bằng mét, được biểu thị bằng cùng màu của chính pixel đó.

Để dễ dàng lưu trữ và sử dụng các mô hình 3D, chúng được lưu dưới dạng tệp “.ply”. Tên là viết tắt của “Polygon File Format” và cấu trúc của chúng bao gồm 3 phần chính:

- Phần thuộc tính: ở đây chúng ta phải khai báo kiểu dữ liệu (float, int32, ...) mô tả từng thuộc tính của một điểm (chẳng hạn như màu sắc và vị trí). Ngoài ra, số lượng đỉnh và mặt của mô hình sẽ được chỉ ra.
- Phần Vertex: ở đây nhóm liệt kê mọi điểm phải được biểu diễn trong không gian 3D cùng với các giá trị thuộc tính của nó. Trong trường hợp này, nhóm phải chỉ ra các vị trí X, Y, Z và màu RGB.
- Phần khuôn mặt: giống như trước, nhưng lần này là khuôn mặt 3D. Không được sử dụng trong trường hợp của nhóm, vì nhóm chỉ đại diện cho các đám mây điểm.

Để đại diện cho đám mây điểm, nhóm chỉ cần sử dụng các vị trí X, Y và Z có được từ quy trình xác định bản đồ độ sâu (disparity map) và nhờ thư viện 'plyfile', chúng tôi chuyển đổi chúng thành một kiểu dữ liệu có thể được chèn chính xác vào bên trong tệp “.ply”.

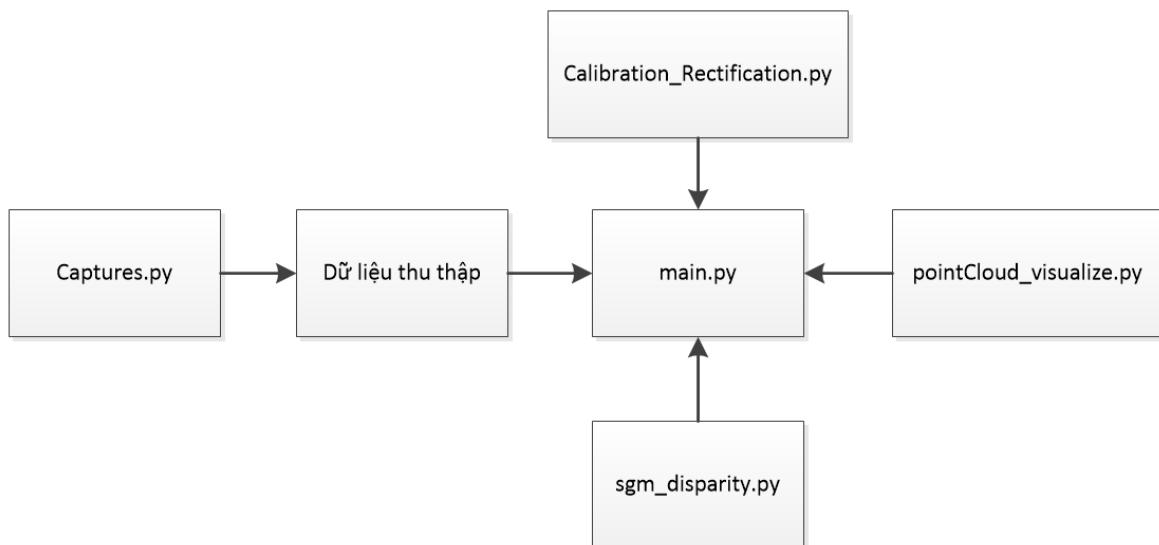


Hình 43 Ví dụ về mô hình 3D dạng .ply

Ngoài ra, nhóm có thể tô màu những điểm đó bằng cách khai thác thực tế là bản đồ chênh lệch của nhóm đã tính toán như một phần bù từ một trong hai hình ảnh, và như vậy nhóm chỉ cần sử dụng các giá trị RGB của hình ảnh 2D, áp dụng chúng cho các giá trị độ sâu được tính toán cho mỗi pixel.

Chương 4: CHƯƠNG TRÌNH VÀ HƯỚNG DẪN SỬ DỤNG

Toàn bộ chương trình có 4 quá trình chính và 1 chương trình chính tổng hợp tất cả (main) và tất cả chương trình đều được viết theo ngôn ngữ lập trình python. Đầu tiên là quá trình thu thập dữ liệu, thứ hai là hiệu chỉnh, tiếp theo là chương trình tính toán độ sâu (disparity) và cuối cùng là chương trình tạo và hiển thị mô hình 3D. Tên các chương và chức năng tương tác thể hiện qua sơ đồ khối sau:



Hình 44 Sơ đồ khối mối liên hệ giữa các chương trình

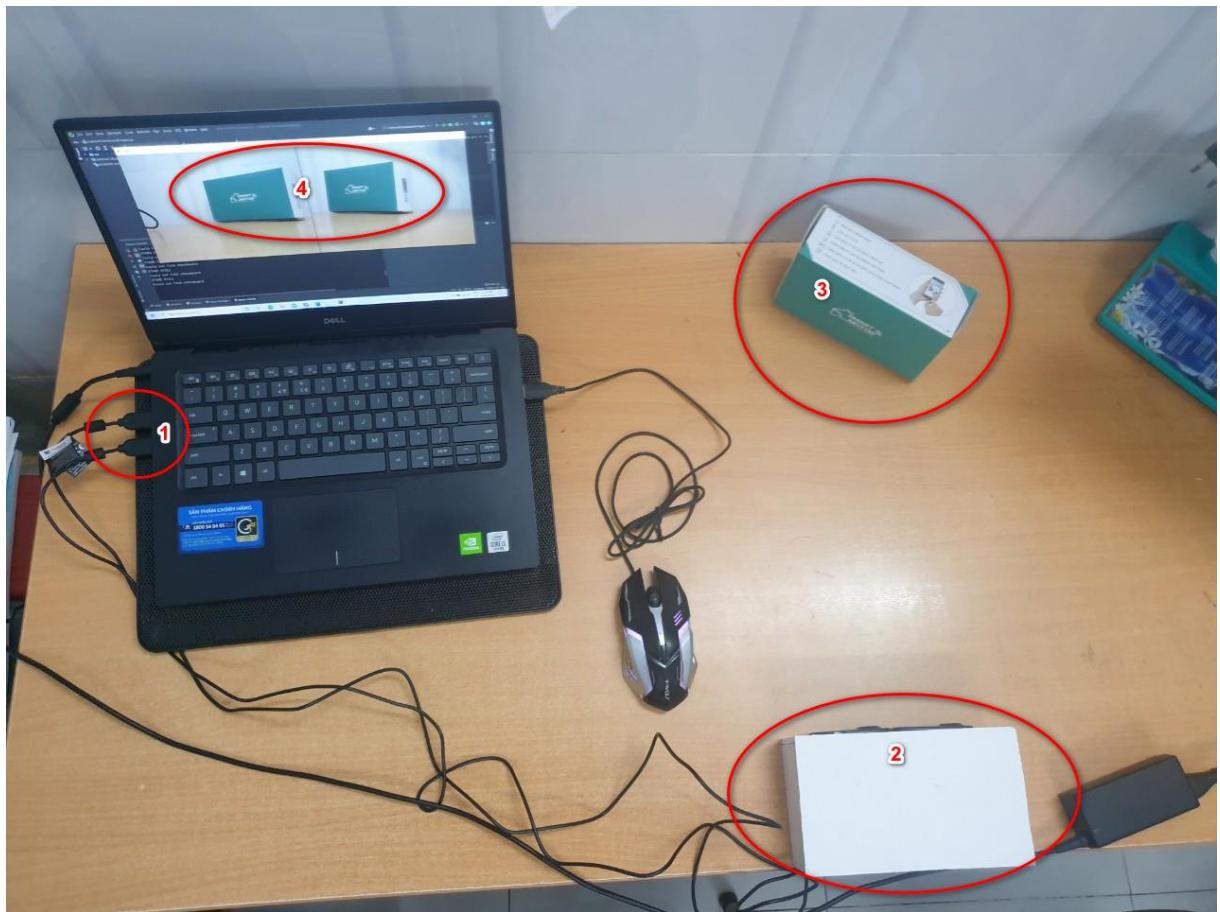
4.1. Chương trình “captures.py”

Đầu tiên ta cần chuẩn bị cặp camera (Hình 45) và thiết lập các thông số quan trọng như:

- Số lượng chessboard cần chụp cho mỗi camera: 20 ảnh.
- Thiết lập cổng kết nối camera trái là 0.
- Thiết lập cổng kết nối camera phải là 1.

Chương trình này kiểm tra nếu chụp 20 tấm ảnh chessboard với kích thước chessboard là 10x7 thì lưu 20 dữ liệu với tên từ 0 → 19, dạng ảnh là png trong thư mục “left” và “right” được chứa trong thư mục “image”.

Sau đó chụp đối tượng ta thu được cặp ảnh trái và phải của đối tượng đó.



Hình 45 Hệ thống kết nối stereo camera với máy tính

Với 1: Kết nối cổng USB từ 2 camera đến máy tính.

2: Stereo camera

3: Đôi tượng (mẫu ví dụ)

4: 2 hình ảnh đối tượng đã được chụp từ stereo camera.

4.2. Chương trình “Calibration_Rectification.py”

Trong chương trình “main” ta gọi chương trình Calibration_Rectification với đầu vào ta phải chọn thư mục chứa 20 ảnh chessboard trái và 20 ảnh chessboard phải. Ta lần lượt thu được các ma trận camera và ma trận Q. Với ma trận Q là:

```

Step 1: Stereo camera is calibrate and rectify
Calibrate camera left ....
Calibrate camera right ....
Stereo calibrate ....
Stereo rectify ....
Done after 3.7683873176574707 seconds
Q ma tran
[[ 1.0000000e+00  0.0000000e+00  0.0000000e+00 -1.57634773e+02]
 [ 0.0000000e+00  1.0000000e+00  0.0000000e+00 -2.50476748e+02]
 [ 0.0000000e+00  0.0000000e+00  0.0000000e+00  8.82764989e+02]
 [ 0.0000000e+00  0.0000000e+00  3.43809206e-01 -0.0000000e+00]]
Step 2: Import and remap 2 stereo image

```

Hình 46 Chạy chương trình Calibration Rectification

Tiếp theo là chọn 2 ảnh vừa được chụp bởi cặp camera, ở đây nhóm chọn ảnh “obj4_left.jpg” và “obj4_right.jpg”.

Sau hiệu chỉnh ta thu được ảnh ở thư mục “remap” với 2 ảnh là “4left.jpg” và “4right.jpg” tức là 2 bức ảnh này đã được hiệu chỉnh.

Cuối cùng là bước cân bằng độ sáng, ở đây nhóm cân bằng theo từng kênh màu RGB và kết quả thu được ở trong thư mục “equalizeHist” và có tên “left4.png” và “right4.png”.

4.3. Chương trình “sgm_disparity.py”

Chương trình tính toán disparity “sgm_disparity.py” cũng được chạy trong chương trình “main.py” sau quá trình cân bằng sáng thì chương trình sẽ chạy như Hình 47. Hình ảnh thu được từ bản đồ độ sâu có tên “left_4disparity_map.png” và ma trận disparity được lưu trong file “4sgm_disparity_matrix.npy”.

```
Step 3: Calculate disparity map

Loading images...

Starting cost computation...
    Computing left and right census... (done in 22.13s)
    Computing cost volumes... (done in 26.02s)

Starting left aggregation computation...
    Processing paths east and west... (done in 17.06s)
    Processing paths south-east and north-west... (done in 17.76s)
    Processing paths south and north... (done in 17.34s)
    Processing paths south-west and north-east... (done in 17.55s)

Selecting best disparities...

Applying median filter...

Fin.

Total execution time = 118.15s
```

Hình 47 Quá trình tính toán disparity

4.4. Chương trình “pointCloud_visualize.py”

Có 2 cách để chạy chương trình tạo đám mây điểm “pointCloud_visualize.py”:

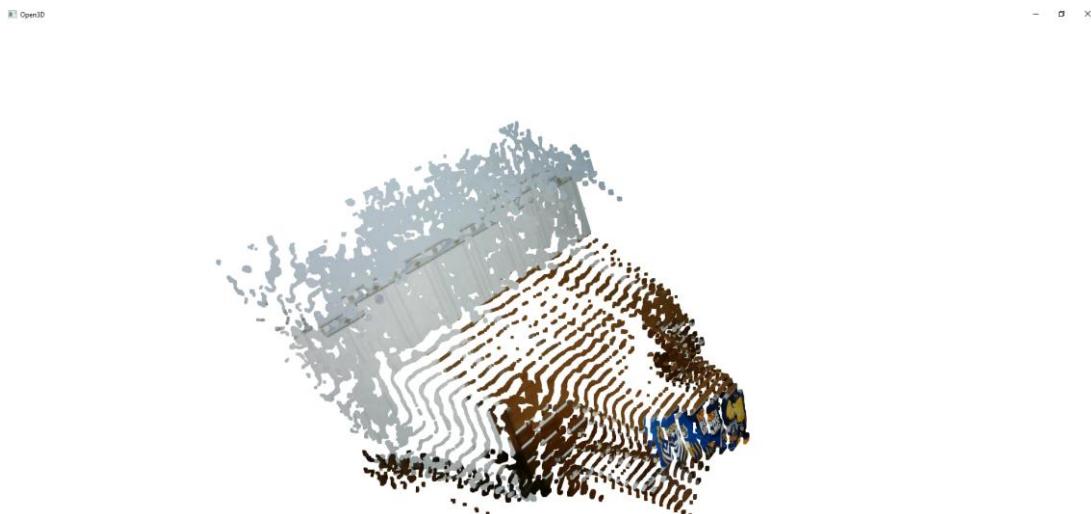
- Chạy cùng chương trình chính “main.py”

```
Step 4: Visualize point cloud  
.\\ply\\pointcloud4.ply saved  
Done
```

Hình 48 Chạy chương trình pointCloud_visualize.py

- Nếu muốn xem lại kết quả của bất kì đối tượng nào đã lưu trước đó thì mở chương trình “pointCloud_visualize.py” và chạy nó và chỉnh nội dung có tên “filename” thành tên khác mà trước đó đã tạo đám mây điểm ảnh và lưu trong thư mục “ply”.

Cả 2 cách đều tương tự nhau và cho ra mô hình đám mây như nhau. Đầu vào của chương trình này là ma trận disparity đã tạo ở mục 4.3, ảnh đã được hiệu chỉnh và ma trận Q lấy được ở mục 4.2. Kết quả trả về là định dạng “.ply” với các nội dung bên trong đã được đề cập ở mục 3.4. Giao diện mô phỏng được thể hiện như hình bên dưới.



Hình 49 Giao diện cửa sổ để xem mô hình 3D của vật thể.

Chương 5: THỰC NGHIỆM VÀ KẾT QUẢ THU ĐƯỢC

Sau thời gian nghiên cứu và thực nghiệm các kết quả thu được:

- Thiết kế và viết chương trình cho giai đoạn thu thập dữ liệu.
- Viết chương trình cho giai đoạn hiệu chỉnh, tính toán disparity, hiển thị mô hình 3D.
- Đã thu thập được dữ liệu từ stereo camera.
- Đã ghép được các đám mây điểm lại với nhau.
- Đã hiển thị đám mây điểm

Đầu tiên nhóm đã thực hiện lại các bước đề cập bên trên và đã thử áp dụng nhiều phương pháp hiệu chỉnh, tính toán disparity sao cho thu được kết quả tốt nhất. Kết quả được thể hiện như sau:

5.1. Thực nghiệm 1: Kết quả tốt nhất nhóm đạt được.

- Hai ảnh đầu vào, để hạn chế ảnh hưởng trực tiếp của ánh sáng mặt trời nên nhóm đã chọn thời điểm thích hợp như buổi sáng, trời râm để thực nghiệm.

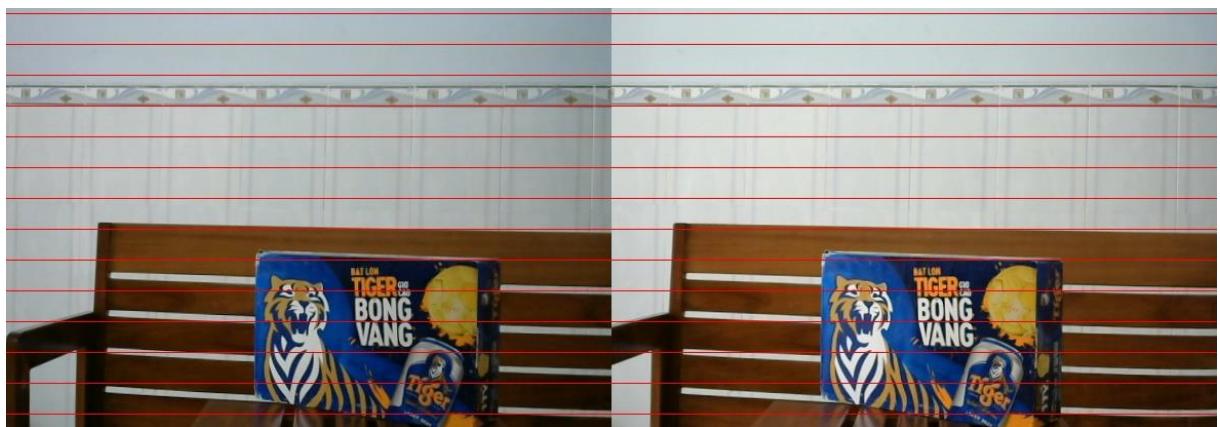


Hình 50 Ảnh trái



Hình 51 Ảnh phải

- Kết quả hiệu chỉnh méo, cân bằng sáng:



Hình 52 So sánh ảnh sau khi hiệu chỉnh



Hình 53 Ảnh đã cân bằng sáng

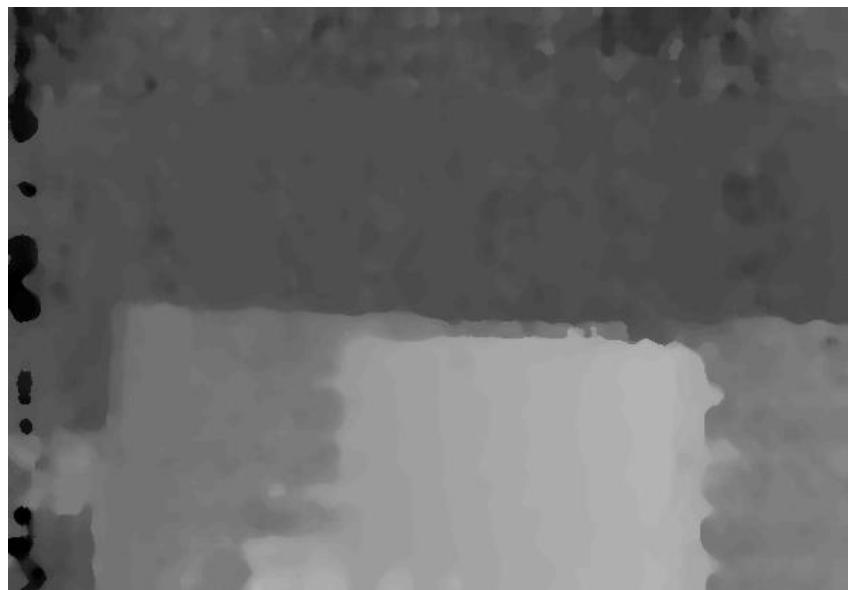
Kết quả hiệu chỉnh và cân bằng sáng mà nhóm thu được khá tốt. Sau đó nhóm đã tìm ra phương pháp semi global matching để tính toán disparity.

Sau khi dùng phương pháp semi global matching nhóm đã thu được kết quả như sau:



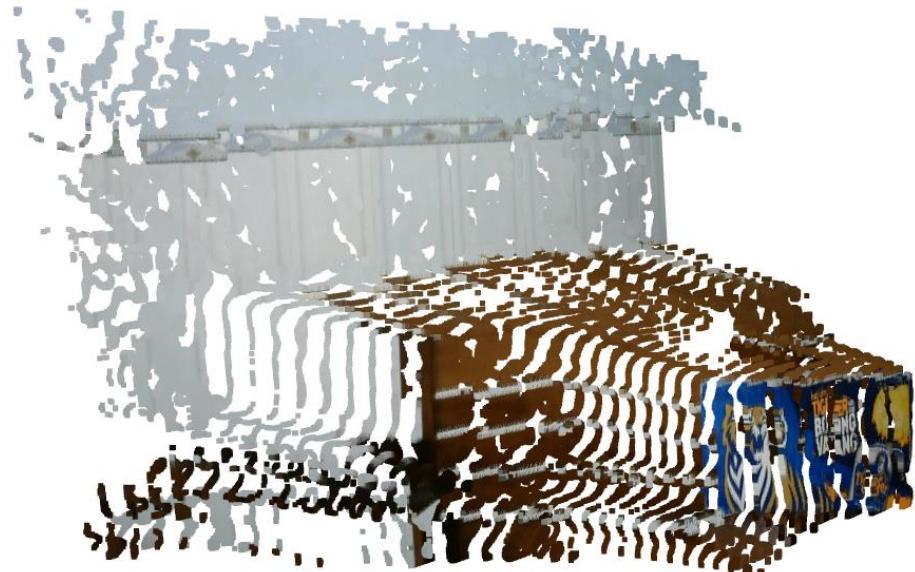
Hình 54 Disparity trước khi lọc trung vị

Vấn đề mà nhóm nhận thấy các vùng khoanh tròn màu đỏ trên bị nhận sai là các giá trị xa quá hoặc gần quá so với thực tế, vì vậy nhóm đã dùng thêm một phương pháp lọc trung vị để lọc thì thu được kết quả rõ ràng hơn.



Hình 55 Disparity thể hiện độ sâu ảnh.

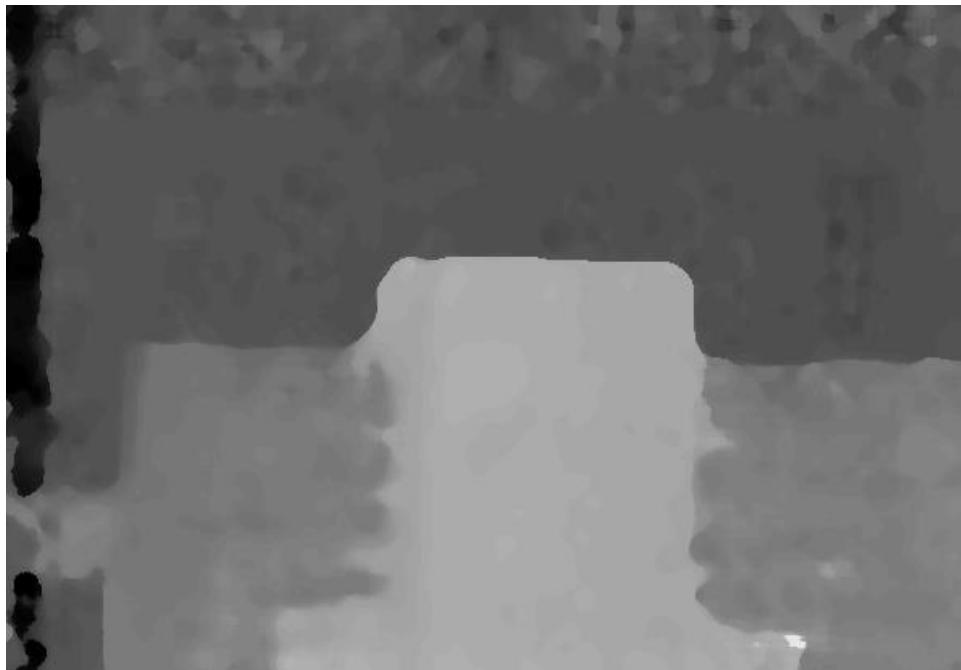
Tiếp theo là chuyển về bản đồ độ sâu thực tế (real depth) dựa trên thông số Q và ma trận disparity. Và từ đó kết hợp với ảnh màu thu được mô hình 3D như sau:



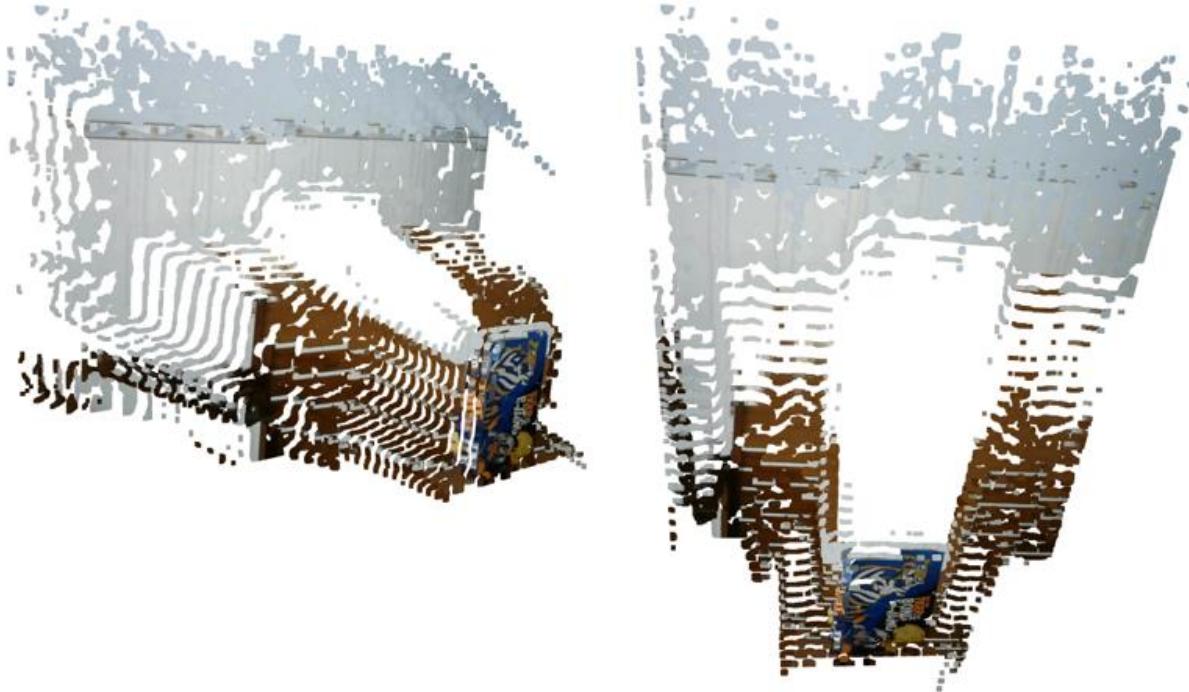
Hình 56 Mô hình 3D của vật thể

Theo kết quả thu được, nhóm đã thu được kết quả khá tốt, mô tả được độ sâu giữa các đối tượng và màu sắc phù hợp. Đây là kết quả tốt nhất mà nhóm thu được sau nhiều lần kiểm tra và so sánh.

5.2. Thực nghiệm 2: Kết quả cùng phương pháp, cùng độ sáng nhưng vật thể khác.



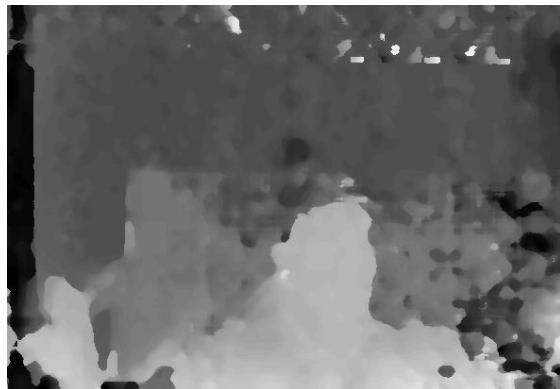
Hình 57 Ánh disparity thực nghiệm 2



Hình 58 Dám mây điểm ảnh thực nghiệm 2

Nhận xét: Ở trường hợp cùng phương pháp, cùng thời điểm ánh sáng và cùng không gian nhóm cũng thu được kết quả khá tốt.

5.3. Thực nghiệm 3: Kết quả cùng phương pháp, khác độ sáng, khác vật thể.



Hình 59 Disparity vật thể thực nghiệm 3



Hình 60 Đám mây điểm ảnh thực nghiệm 3

Nhận xét: Ở thời điểm khác độ sáng, tuy nhóm cũng thu được kết quả thể hiện trước sau, màu sắc cũng thu được tương đối nhưng giữa các đối tượng kết quả không được tốt.

5.4. Thực nghiệm 4: Kết quả cùng phương pháp, khác độ sáng, khác vật thể.



Hình 61 Disparity vật thể thực nghiệm 4



Hình 62 Dám mây điểm ảnh thực nghiệm 4

Nhận xét: Ở cùng trường hợp với thực nghiệm 3, tuy nhiên vật thể to hơn, việc xử lý ảnh được tốt hơn nên thu được đám mây điểm ảnh tương đối tốt.

5.5. Thực nghiệm 5: Kết quả so sánh phương pháp tìm bản đồ chênh lệch (disparity maps):



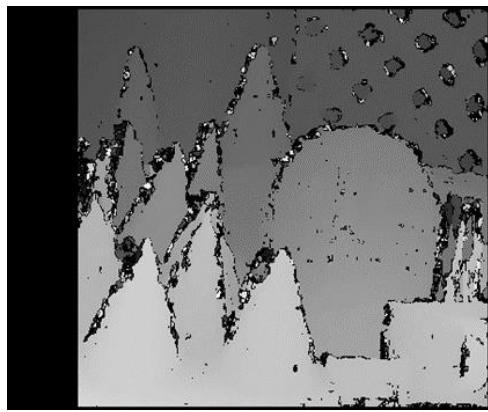
Hình 63 Ảnh cones trái.



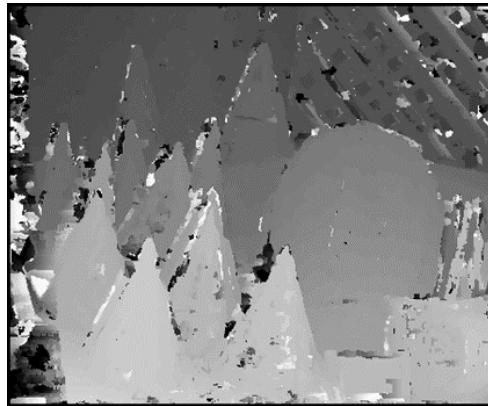
Hình 64 Ảnh cones phải



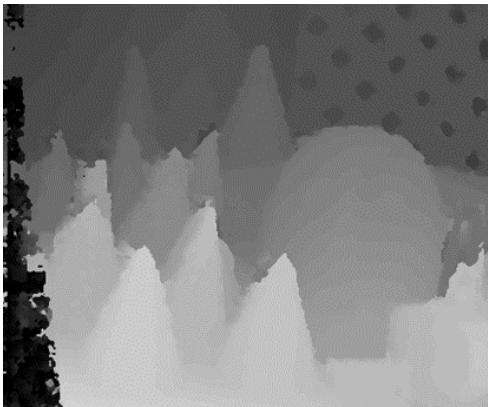
Hình 65 Groud truth



Hình 66 Phương pháp Block matching opencv



Hình 67 Disparity phương pháp SSD stereo matching



Hình 68 Disparity phương pháp semi-global matching

Phương pháp	Block matching open cv	SSD stereo matching	Semi-global matching
K	70.77%	76.45%	86.47%

Với K: độ tốt bản đồ chênh lệch của các phương pháp so với bản đồ chênh lệch thực.

Nhận xét: Khi tính được bản đồ chênh lệch cho kết quả tốt thì việc xuất point cloud cho kết quả tốt là tương đương, do đó với kết quả ở bảng so sánh trên thì nhóm đã sử dụng thuật toán semi global matching cho kết quả tốt hơn.

KẾT LUẬN VÀ KIẾN NGHỊ

Kết luận

Kết quả của đề tài mang lại nhiều kết quả xác thực với nhiệm vụ đã đặt ra và tương đồng với những nghiên cứu của các tác giả trên thế giới, như:

- Đã xây dựng lại một cơ sở lý thuyết về thị giác nổi rõ ràng
- Những điểm tương đồng đã được tìm ra tọa độ 3D và biểu diễn thành ảnh 3D để phân biệt chiều sâu của sản phẩm
- Thuật toán semi global matching được vận dụng để khôi phục lại ảnh nổi chênh lệch độ sâu của sản phẩm theo toàn cục, ảnh 3D mô tả chiều sâu của sản phẩm thông qua chỉ số độ sáng của màu sắc
- Nhiều hình ảnh minh họa sinh động, đề tài trình bày một cách tường minh, là tài liệu tham khảo hữu ích làm cơ sở cho những nghiên cứu tiếp theo về thị giác nổi

Tuy nhiên vẫn còn nhiều hạn chế:

- Ở thời điểm khác nhau, cường độ sáng khác nhau thì sẽ thu được kết quả không chính xác.
- Kích thước đối tượng cũng là vấn đề ảnh hưởng tới tính toán bản đồ độ sâu.

Hướng phát triển

Đề tài cần được nghiên cứu phát triển thực hiện được trong thời gian thực để vận dụng vào sản xuất như có thể chế tạo máy đo kích thước 3D của sản phẩm hay là thiết bị giám sát sản phẩm trên băng tải, cung cấp tọa độ và kích thước sản phẩm để điều khiển cánh tay robot gấp sản phẩm ra khỏi băng tải. Các thuật toán tìm sự phù hợp tương đồng ảnh nổi cần được cải tiến để cho ra kết quả pixel tương đồng chính xác nhất. Thuật toán của nhóm tìm hiểu đã thu được phân biệt độ chênh lệch rõ ràng nhưng còn chậm muộn thu được kết quả thực phải cần có GPU mạnh.

Đồng thời tăng thêm góc nhìn bằng cách sử dụng nhiều cặp stereo camera để xử lý và cho ra hình dạng, kích thước và các chi tiết cụ thể của vật thể, hoặc làm giá xoay cho vật thể.

TÀI LIỆU THAM KHẢO

- [1] A. Klaus, M. Sormann, and K. Karner, “Segment-based stereo matching using belief propagation and a self-adapting dissimilarity measure,” *Proc. - Int. Conf. Pattern Recognit.*, vol. 3, no. January, pp. 15–18, 2006, doi: 10.1109/ICPR.2006.1033.
- [2] Z. Murez, T. van As, J. Bartolozzi, A. Sinha, V. Badrinarayanan, and A. Rabinovich, “Atlas: End-to-End 3D Scene Reconstruction from Posed Images,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 12352 LNCS, pp. 414–431, 2020, doi: 10.1007/978-3-030-58571-6_25.
- [3] P. H. Lee, J. W. Huang, and H. Y. Lin, “3D model reconstruction based on multiple view image capture,” *ISPACS 2012 - IEEE Int. Symp. Intell. Signal Process. Commun. Syst.*, no. Ispacs, pp. 58–63, 2012, doi: 10.1109/ISPACS.2012.6473453.
- [4] B. Ummenhofer and T. Brox, “Point-based 3D reconstruction of thin objects,” *Proc. IEEE Int. Conf. Comput. Vis.*, pp. 969–976, 2013, doi: 10.1109/ICCV.2013.124.
- [5] W. W. Ma, M. H. Le, and K. H. Jo, “3D reconstruction and measurement of indoor object using stereo camera,” *Proc. 6th Int. Forum Strateg. Technol. IFOST 2011*, vol. 2, no. July 2015, pp. 738–742, 2011, doi: 10.1109/IFOST.2011.6021128.
- [6] C. Teutsch, D. Berndt, A. Sobotta, and S. Sperling, “A flexible photogrammetric stereo vision system for capturing the 3D shape of extruded profiles,” *Two-Three-Dimensional Methods Insp. Metrol. IV*, vol. 6382, no. November, p. 63820M, 2006, doi: 10.1117/12.685546.
- [7] J. Bigun, *Vision with Direction*. 2006.
- [8] Ts Lê Mỹ Hà, *Giáo trình thị giác máy tính và ứng dụng*. Nhà xuất bản đại học quốc gia TP.Hồ Chí Minh., 2019.
- [9] Ts. Lê Mỹ Hà, *Giáo trình Máy và hệ thống xử lý ảnh trong công nghiệp*. Nhà xuất bản đại học quốc gia TP. Hồ Chí Minh, 2017.
- [10] D. Hafner, O. Demetz, and J. Weickert, “Why is the census transform good for robust optic flow computation?,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 7893 LNCS, pp. 210–221, 2013, doi: 10.1007/978-3-642-38267-3_18.
- [11] R. Spangenberg, T. Langner, and R. Rojas, “Weighted semi-global matching and center-symmetric census transform for robust driver assistance,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8048 LNCS, no. PART 2, pp. 34–41, 2013, doi: 10.1007/978-3-642-40246-3_5.
- [12] J. Ko and Y. S. Ho, “Stereo matching using census transform of adaptive window sizes with gradient images,” *2016 Asia-Pacific Signal Inf. Process. Assoc. Annu. Summit Conf. APSIPA 2016*, pp. 2–5, 2017, doi: 10.1109/APSIPA.2016.7820827.
- [13] “Semi-global matching.” https://en.wikipedia.org/wiki/Semi-global_matching.

- [14] H. Hirschmüller, “Stereo processing by semiglobal matching and mutual information,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 30, no. 2, pp. 328–341, 2008, doi: 10.1109/TPAMI.2007.1166.

PHỤ LỤC

Chương trình main

```
import Calibration_Rectification  
  
import sgm_disparity  
  
import pointCloud_visualize  
  
import cv2  
  
chessboard_path_left = ".\\image\\left2\\*.png"  
  
chessboard_path_right = ".\\image\\right2\\*.png"  
  
# list image reference  
  
imgL1 = ".\\image\\obj3_left.jpg"  
  
imgR1 = ".\\image\\obj3_right.jpg"  
  
imgL2 = ".\\image\\obj4_left.jpg"  
  
imgR2 = ".\\image\\obj4_right.jpg"  
  
imgL3 = ".\\image\\captures8\\left\\0.png"  
  
imgR3 = ".\\image\\captures8\\right\\0.png"  
  
imgL = [cv2.imread(imgL1), cv2.imread(imgL2), cv2.imread(imgL3)]  
  
imgR = [cv2.imread(imgR1), cv2.imread(imgR2), cv2.imread(imgR3)]  
  
img_choose = 1 # value to choose stereo image of list imgL/imgR above  
  
filename = "4" # change filename to save new data, filename is lastname for 2 image  
left/right  
  
print("Step 1: Stereo camera is calibrate and retify")  
  
mapLX, mapLY, mapRX, mapRY, Q =  
Calibration_Rectification.calibrate_rectify(chessboard_path_left,  
                                              chessboard_path_right,  
                                              show=False)  
  
print("Q ma tran")
```

```
print(Q)

print("Step 2: Import and remap 2 stereo image ")

dstL = cv2.remap(imgL[img_choose], mapLX, mapLY, cv2.INTER_LINEAR,
cv2.BORDER_CONSTANT)

dstR = cv2.remap(imgR[img_choose], mapRX, mapRY, cv2.INTER_LINEAR,
cv2.BORDER_CONSTANT)

dstL, dstR = Calibration_Rectification.show_Image_remap(imgL[img_choose],
imgR[img_choose],
dstL, dstR, filename=filename)

# Calibration_Rectification.draw_comparison(dstL, dstR, filename)

dstL = Calibration_Rectification.equalizeHist_RGB(dstL, 'left'+filename+'.png')

dstR = Calibration_Rectification.equalizeHist_RGB(dstR, 'right'+filename+'.png')

print("Step 3: Calculate disparity map ")

parameters = sgm_disparity.Parameters(max_disparity=64, P1=10, P2=120, cscale=(7,
7), bsize=(11, 11))

paths = sgm_disparity.Paths()

save_images = True

left_name = '.\\remap\\' + filename + 'left.jpg'

right_name = '.\\remap\\' + filename + 'right.jpg'
```

```
disparity = sgm_disparity.sgm_disparity(left_name, right_name, filename, parameters,
paths, save_images)

cv2.imshow('disparity', disparity)

cv2.waitKey(0)

cv2.destroyAllWindows()

print("Step 4: Visualize point cloud")

img = cv2.imread(left_name)

pointCloud_visualize.create_pointCloud(disparity, Q, img, filename)

pointCloud_visualize.visualize_plyfile('.\\ply\\pointcloud' + filename + '.ply')
```

Chương trình captures chessboard

```
import time

import cv2

MAX_FRAMES_CHECKBOARD = 20

chessboardCaptured = 0

left_camera_port = 0

right_camera_port = 1

left_camera = cv2.VideoCapture(left_camera_port)

time.sleep(0.5)

right_camera = cv2.VideoCapture(right_camera_port)

time.sleep(0.5)
```

```

# chessboard image capturing cycle

while chessboardCaptured != MAX_FRAMES_CHECKBOARD:

    cv2.waitKey(2000)

    print('STAND STILL')

    cv2.waitKey(1000)

# Capturing image from camera

left_return_value, leftImage = left_camera.read()

right_return_value, rightImage = right_camera.read()

cv2.imshow('left', leftImage)

cv2.imshow('right', rightImage)

# Finding chessboard corners

left_gray = cv2.cvtColor(leftImage, cv2.COLOR_BGR2GRAY)

right_gray = cv2.cvtColor(rightImage, cv2.COLOR_BGR2GRAY)

left_ret, leftCorners = cv2.findChessboardCorners(left_gray, (9, 6), None)

rightRet, rightCorners = cv2.findChessboardCorners(right_gray, (9, 6), None)

# only save images where the chessboard was found

if left_ret and rightRet:

    cv2.imwrite("./calibration/left/" + str(chessboardCaptured) + ".png", leftImage)

    cv2.imwrite("./calibration/right/" + str(chessboardCaptured) + ".png", rightImage)

```

```

cv2.waitKey(1000)

chessboardCaptured = chessboardCaptured + 1

print("OK, " + str(MAX_FRAMES_CHECKBOARD - chessboardCaptured) + " more
images needed")

else:

    print("Could not find chessboard")


del left_camera

del right_camera

cv2.destroyAllWindows()

```

Chương trình captures

```

import cv2

import time

left_camera_port = 0

right_camera_port = 1


left_camera = cv2.VideoCapture(left_camera_port)

time.sleep(0.5)

right_camera = cv2.VideoCapture(right_camera_port)

time.sleep(0.5)


# MAIN CAPTURING LOOP, storing images in the folder

counter = 0

while counter < 1

```

```

cv2.waitKey(1000)

print('Capturing...')

# Capturing image from camera

left_return_value, left_image = left_camera.read()

right_return_value, right_image = right_camera.read()

cv2.imwrite('./remap/captures/left/' + str(counter) + '.png', left_image)

cv2.imwrite('./remap/captures/right/' + str(counter) + '.png', right_image)

counter += 1

```

Chương trình Calibration_Retification

```

import numpy as np

import cv2

import glob

import time

def Individual_calibrate(nx, ny, basepath, show_corners):

    # termination criteria for function : cornerSubpix

    criteria = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30, 0.001)

    chessboard_flags = cv2.CALIB_CB_ADAPTIVE_THRESH + cv2.CALIB_CB_FAST_CHECK
+ cv2.CALIB_CB_NORMALIZE_IMAGE

    objp = np.zeros((nx * ny, 3), np.float32) # points in real world space

    objp[:, :2] = np.mgrid[0:nx, 0:ny].T.reshape(-1, 2)

```

```

# Arrays to store object points and image points from all the images.

imgpoints = [] # 2d points in image plane.

objpoints = [] # return data table consist of model plane and image plane

# Make a list of calibration images

images = glob.glob(basepath)

gray = []

# Step through the list and search for chessboard corners

for fname in range(0, len(images)):

    img = cv2.imread(images[fname])

    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)

    # Find the chessboard corners

    ret, corners = cv2.findChessboardCorners(gray, (nx, ny), chessboard_flags)

    # If found, add object points, image points

    if ret:

        objpoints.append(objp)

        corners2 = cv2.cornerSubPix(gray, corners, (11, 11), (-1, -1), criteria)

        imgpoints.append(corners)

        if show_corners:

            cv2.drawChessboardCorners(img, (nx, ny), corners2, ret)

            cv2.imshow('img', img)

            cv2.waitKey(500)

```

```

if show_corners:

    cv2.destroyAllWindows()

    size = gray.shape[::-1]

    ret, mtx, dist, rvecs, tvecs = cv2.calibrateCamera(objpoints, imgpoints, size, None,
None)

    return objpoints, imgpoints, size, ret, mtx, dist, rvecs, tvecs


def Re_projection_Error(objpoints, imgpoints, rvecs, tvecs, mtx, dist):

    mean_error = 0

    for i in range(len(objpoints)):

        imgpoints2, _ = cv2.projectPoints(objpoints[i], rvecs[i], tvecs[i], mtx, dist)

        error = cv2.norm(imgpoints[i], imgpoints2, cv2.NORM_L2) / len(imgpoints2)

        mean_error += error

    return mean_error / len(objpoints)


def draw_comparison(im_l, im_r, filename):

    # Draw horizontal lines on images

    h, w, c = im_l.shape

    print("h", h)

    print("w", w)

    for j in range(5, h, 30):

        cv2.line(im_l, (0, j), (w, j), (0, 0, 255), 1)

        cv2.line(im_r, (0, j), (w, j), (0, 0, 255), 1)

    # create unique image corresponding to the alignment of left and right

    aligned = np.hstack((im_l, im_r))

```

```

cv2.imwrite('.\\remap\\' + filename + 'comparison.jpg', aligned)

def calibrate_rectify(path_left, path_right, show):
    """
    :param path_left: folder path left image chessboard
    :param path_right: folder path right image chessboard
    :param show: boolean value to show image loading
    :return: matrix x/y of the undistorted rectified stereo image to remap
    """

    start_time = time.time()

    """ Step 1: Individual calibration of the right and left cameras of the stereo setup """
    print("    Calibrate camera left ....")
    _, img_ptsL, _, ret_left, mtxL, distL, rvecsL, tvecsL = Individual_calibrate(nx=9,
        ny=6,
        basepath=path_left,
        show_corners=show)

    print("    Calibrate camera right ....")
    obj_pts, img_ptsR, gray_size, ret_right, mtxR, distR, rvecsR, tvecsR =
    Individual_calibrate(nx=9,
        ny=6,
        basepath=path_right,
        show_corners=show)

    # print(Re_projection_Error(obj_pts, img_ptsR, rvecsR, tvecsR, mtxR, distR))
    # print(Re_projection_Error(obj_pts, img_ptsL, rvecsL, tvecsL, mtxL, distL))

```

```

"""\ Step 2: Performing stereo calibration with fixed intrinsic parameters """"

flags = 0

# flags |= cv2.CALIB_FIX_INTRINSIC

flags |= cv2.CALIB_USE_INTRINSIC_GUESS

flags |= cv2.CALIB_FIX_FOCAL_LENGTH


flags |= cv2.CALIB_ZERO_TANGENT_DIST

# Here we fix the intrinsic camara matrixes so that only Rot, Trns, Emat and Fmat are
calculated.

# Hence intrinsic parameters are the same


criteria_stereo = (cv2.TERM_CRITERIA_EPS + cv2.TERM_CRITERIA_MAX_ITER, 30,
0.001)

# This step is performed to transformation between the two cameras and calculate
Essential and Fundamenatl matrix

print("      Stereo calibrate ....")

retS, new_mtxL, distL, new_mtxR, distR, Rot, Trns, Emat, Fmat =
cv2.stereoCalibrate(obj_pts, img_ptsL, img_ptsR,
                     mtxL, distL, mtxR, distR,
                     gray_size,
                     criteria_stereo, flags)

"""\ Step 3: Stereo Rectification """

```

```

print("    Stereo rectify ....")

rectify_scale = 1

rect_l, rect_r, proj_mat_l, proj_mat_r, Q, roiL, roiR = cv2.stereoRectify(new_mtxL,
distL, new_mtxR, distR,

                                gray_size,
                                Rot, Trns, rectify_scale, (0, 0))

"""

Step 4: Compute the mapping required to obtain the undistorted rectified stereo
image pair """

Left_Stereo_Map_X, Left_Stereo_Map_Y = cv2.initUndistortRectifyMap(new_mtxL,
distL, rect_l, proj_mat_l,
                                gray_size, cv2.CV_16SC2)

Right_Stereo_Map_X, Right_Stereo_Map_Y =
cv2.initUndistortRectifyMap(new_mtxR, distR, rect_r, proj_mat_r,
                                gray_size, cv2.CV_16SC2)

time_finish = time.time() - start_time

print("    Done after %s seconds " % time_finish)

return Left_Stereo_Map_X, Left_Stereo_Map_Y, Right_Stereo_Map_X,
Right_Stereo_Map_Y, Q

def show_Image_remap(imgL, imgR, dstL, dstR, filename):

dstR = dstR[40:450, 20: 610]

dstL = dstL[40:450, 20: 610]

imgL = imgL[40:450, 20: 610]

```

```

imgR = imgR[40:450, 20: 610]

hor1 = np.concatenate((imgL, imgR), axis=1)

hor2 = np.concatenate((dstL, dstR), axis=1)

remap_result = np.concatenate((hor1, hor2), axis=0)

remap_result = cv2.resize(remap_result, (885, 615), None, 1, 1)

cv2.imwrite('.\\remap\\' + filename + 'left.jpg', dstL)

cv2.imwrite('.\\remap\\' + filename + 'right.jpg', dstR)

cv2.imshow('remap result', remap_result)

cv2.waitKey(0)

cv2.destroyAllWindows()

return dstL, dstR

def equalizeHist_RGB(imgRGB, filename):

    R, G, B = cv2.split(imgRGB)

    output1_R = cv2.equalizeHist(R)

    output1_G = cv2.equalizeHist(G)

    output1_B = cv2.equalizeHist(B)

    equ = cv2.merge((output1_R, output1_G, output1_B))

    cv2.imwrite('.\\equalizeHist\\'+filename, equ)

    return equ

```

Chương trình sgm_disparity

```
import argparse
```

```
import sys

import time as t

import cv2

import numpy as np


class Direction:

    def __init__(self, direction=(0, 0), name='invalid'):

        """



        represent a cardinal direction in image coordinates (top left = (0, 0) and bottom right = (1, 1)).

        :param direction: (x, y) for cardinal direction.

        :param name: common name of said direction.

        """



        self.direction = direction

        self.name = name



# 8 defined directions for sgm

N = Direction(direction=(0, -1), name='north')

NE = Direction(direction=(1, -1), name='north-east')

E = Direction(direction=(1, 0), name='east')

SE = Direction(direction=(1, 1), name='south-east')

S = Direction(direction=(0, 1), name='south')

SW = Direction(direction=(-1, 1), name='south-west')

W = Direction(direction=(-1, 0), name='west')

NW = Direction(direction=(-1, -1), name='north-west')
```

```
class Paths:
```

```
    def __init__(self):
```

```
        """
```

```
        represent the relation between the directions.
```

```
        """
```

```
        self.paths = [N, NE, E, SE, S, SW, W, NW]
```

```
        self.size = len(self.paths)
```

```
        self.effective_paths = [(E, W), (SE, NW), (S, N), (SW, NE)]
```

```
class Parameters:
```

```
    def __init__(self, max_disparity=64, P1=5, P2=70, csizen=(7, 7), bsize=(3, 3)):
```

```
        """
```

```
        represent all parameters used in the sgm algorithm.
```

```
:param max_disparity: maximum distance between the same pixel in both images.
```

```
:param P1: penalty for disparity difference = 1
```

```
:param P2: penalty for disparity difference > 1
```

```
:param csizen: size of the kernel for the census transform.
```

```
:param bsize: size of the kernel for blurring the images and median filtering.
```

```
        """
```

```
        self.max_disparity = max_disparity
```

```
        self.P1 = P1
```

```
        self.P2 = P2
```

```
        self.csizen = csizen
```

```
        self.bsize = bsize
```

```
def load_images(left_name, right_name, parameters):
```

```
....  
  
read and blur stereo image pair.  
  
:param left_name: name of the left image.  
  
:param right_name: name of the right image.  
  
:param parameters: structure containing parameters of the algorithm.  
  
:return: blurred left and right images.  
  
....  
  
left = cv2.imread(left_name, 0)  
  
left = cv2.GaussianBlur(left, parameters.bsize, 0, 0)  
  
cv2.imwrite('.\\blur\\left_blur.jpg', left)  
  
right = cv2.imread(right_name, 0)  
  
right = cv2.GaussianBlur(right, parameters.bsize, 0, 0)  
  
cv2.imwrite('.\\blur\\right_blur.jpg', left)  
  
return left, right
```

```
def get_indices(offset, dim, direction, height):  
  
....  
  
for the diagonal directions (SE, SW, NW, NE), return the array of indices for the  
current slice.  
  
:param offset: difference with the main diagonal of the cost volume.  
  
:param dim: number of elements along the path.  
  
:param direction: current aggregation direction.  
  
:param height: H of the cost volume.  
  
:return: arrays for the y (H dimension) and x (W dimension) indices.  
  
....  
  
y_indices = []
```

```

x_indices = []

for i in range(0, dim):

    if direction == SE.direction:

        if offset < 0:

            y_indices.append(-offset + i)

            x_indices.append(0 + i)

        else:

            y_indices.append(0 + i)

            x_indices.append(offset + i)

    if direction == SW.direction:

        if offset < 0:

            y_indices.append(height + offset - i)

            x_indices.append(0 + i)

        else:

            y_indices.append(height - i)

            x_indices.append(offset + i)

return np.array(y_indices), np.array(x_indices)

```

def get_path_cost(slice_, offset, parameters):

....

part of the aggregation step, finds the minimum costs in a D x M slice (where M = the number of pixels in the

```

given direction)

:param slice_: M x D array from the cost volume.

:param offset: ignore the pixels on the border.

:param parameters: structure containing parameters of the algorithm.

:return: M x D array of the minimum costs for a given slice in a given direction.

"""

other_dim = slice_.shape[0]

disparity_dim = slice_.shape[1]

disparities = [d for d in range(disparity_dim)] * disparity_dim

disparities = np.array(disparities).reshape(disparity_dim, disparity_dim)

penalties = np.zeros(shape=(disparity_dim, disparity_dim), dtype=slice_.dtype)

penalties[np.abs(disparities - disparities.T) == 1] = parameters.P1

penalties[np.abs(disparities - disparities.T) > 1] = parameters.P2

minimum_cost_path = np.zeros(shape=(other_dim, disparity_dim),
                             dtype=slice_.dtype)

minimum_cost_path[offset - 1, :] = slice_[offset - 1, :]

for i in range(offset, other_dim):

    previous_cost = minimum_cost_path[i - 1, :]

    current_cost = slice_[i, :]

    costs = np.repeat(previous_cost, repeats=disparity_dim,
                      axis=0).reshape(disparity_dim, disparity_dim)

    costs = np.amin(costs + penalties, axis=0)

```

```

minimum_cost_path[i, :] = current_cost + costs - np.amin(previous_cost)

return minimum_cost_path


def aggregate_costs(cost_volume, parameters, paths):
    """
    second step of the sgm algorithm, aggregates matching costs for N possible
    directions (8 in this case).

    :param cost_volume: array containing the matching costs.

    :param parameters: structure containing parameters of the algorithm.

    :param paths: structure containing all directions in which to aggregate costs.

    :return: H x W x D x N array of matching cost for all defined directions.

    """

    height = cost_volume.shape[0]
    width = cost_volume.shape[1]
    disparities = cost_volume.shape[2]
    start = -(height - 1)
    end = width - 1

    aggregation_volume = np.zeros(shape=(height, width, disparities, paths.size),
                                   dtype=cost_volume.dtype)

    path_id = 0
    for path in paths.effective_paths:
        print('\tProcessing paths {} and {}'.format(path[0].name, path[1].name), end='')
        sys.stdout.flush()

```

```

dawn = t.time()

main_aggregation = np.zeros(shape=(height, width, disparities),
                           dtype=cost_volume.dtype)

opposite_aggregation = np.copy(main_aggregation)

main = path[0]

if main.direction == S.direction:

    for x in range(0, width):

        south = cost_volume[0:height, x, :]

        north = np.flip(south, axis=0) # Reverse the order of elements in an array
        along the given axis.

        main_aggregation[:, x, :] = get_path_cost(south, 1, parameters)

        opposite_aggregation[:, x, :] = np.flip(get_path_cost(north, 1, parameters),
        axis=0)

if main.direction == E.direction:

    for y in range(0, height):

        east = cost_volume[y, 0:width, :]

        west = np.flip(east, axis=0)

        main_aggregation[y, :, :] = get_path_cost(east, 1, parameters)

        opposite_aggregation[y, :, :] = np.flip(get_path_cost(west, 1, parameters),
        axis=0)

if main.direction == SE.direction:

    for offset in range(start, end):

        south_east = cost_volume.diagonal(offset=offset).T

```

```

north_west = np.flip(south_east, axis=0)

dim = south_east.shape[0]

y_se_idx, x_se_idx = get_indices(offset, dim, SE.direction, None)

y_nw_idx = np.flip(y_se_idx, axis=0)

x_nw_idx = np.flip(x_se_idx, axis=0)

main_aggregation[y_se_idx, x_se_idx, :] = get_path_cost(south_east, 1,
parameters)

opposite_aggregation[y_nw_idx, x_nw_idx, :] = get_path_cost(north_west, 1,
parameters)

if main.direction == SW.direction:

    for offset in range(start, end):

        south_west = np.flipud(cost_volume).diagonal(offset=offset).T

        north_east = np.flip(south_west, axis=0)

        dim = south_west.shape[0]

        y_sw_idx, x_sw_idx = get_indices(offset, dim, SW.direction, height - 1)

        y_ne_idx = np.flip(y_sw_idx, axis=0)

        x_ne_idx = np.flip(x_sw_idx, axis=0)

        main_aggregation[y_sw_idx, x_sw_idx, :] = get_path_cost(south_west, 1,
parameters)

        opposite_aggregation[y_ne_idx, x_ne_idx, :] = get_path_cost(north_east, 1,
parameters)

aggregation_volume[:, :, :, path_id] = main_aggregation

aggregation_volume[:, :, :, path_id + 1] = opposite_aggregation

path_id = path_id + 2

```

```

dusk = t.time()

print('\t(done in {:.2f}s)'.format(dusk - dawn))

return aggregation_volume

def compute_costs(left, right, parameters, save_images):
    """
    first step of the sgm algorithm, matching cost based on census transform and
    hamming distance.

    :param left: left image.

    :param right: right image.

    :param parameters: structure containing parameters of the algorithm.

    :param save_images: whether to save census images or not.

    :return: H x W x D array with the matching costs.

    """

    assert left.shape[0] == right.shape[0] and left.shape[1] == right.shape[1], 'left & right
must have the same shape.'

    assert parameters.max_disparity > 0, 'maximum disparity must be greater than 0.'


    height = left.shape[0]
    width = left.shape[1]
    cheight = parameters.csizes[0]
    cwidth = parameters.csizes[1]
    y_offset = int(cheight / 2)
    x_offset = int(cwidth / 2)

```

```
disparity = parameters.max_disparity

left_img_census = np.zeros(shape=(height, width), dtype=np.uint8)
right_img_census = np.zeros(shape=(height, width), dtype=np.uint8)
left_census_values = np.zeros(shape=(height, width), dtype=np.uint64)
right_census_values = np.zeros(shape=(height, width), dtype=np.uint64)

print('\tComputing left and right census...', end="")
sys.stdout.flush()

dawn = t.time()

# pixels on the border will have no census values

for y in range(y_offset, height - y_offset):
    for x in range(x_offset, width - x_offset):
        left_census = np.int64(0)
        center_pixel = left[y, x]
        reference = np.full(shape=(cheight, cwidth), fill_value=center_pixel,
                           dtype=np.int64)
        image = left[(y - y_offset):(y + y_offset + 1), (x - x_offset):(x + x_offset + 1)]
        comparison = image - reference
        for j in range(comparison.shape[0]):
            for i in range(comparison.shape[1]):
                if (i, j) != (y_offset, x_offset):
                    left_census = left_census << 1
                    if comparison[j, i] < 0:
                        bit = 1
                    else:
```

```

bit = 0

left_census = left_census | bit

left_img_census[y, x] = np.uint8(left_census)

left_census_values[y, x] = left_census


right_census = np.int64(0)

center_pixel = right[y, x]

reference = np.full(shape=(cheight, cwidth), fill_value=center_pixel,
dtype=np.int64)

image = right[(y - y_offset):(y + y_offset + 1), (x - x_offset):(x + x_offset + 1)]

comparison = image - reference

for j in range(comparison.shape[0]):

    for i in range(comparison.shape[1]):

        if (i, j) != (y_offset, x_offset):

            right_census = right_census << 1

            if comparison[j, i] < 0:

                bit = 1

            else:

                bit = 0

            right_census = right_census | bit

            right_img_census[y, x] = np.uint8(right_census)

            right_census_values[y, x] = right_census


dusk = t.time()

print('\t(done in {:.2f}s)'.format(dusk - dawn))

```

```

if save_images:

    cv2.imwrite('left_census.png', left_img_census)

    cv2.imwrite('right_census.png', right_img_census)

    # cv2.imshow('left_census', left_img_census)

    # cv2.imshow('right_census', right_img_census)


print('\tComputing cost volumes...', end="")

sys.stdout.flush()

dawn = t.time()

left_cost_volume = np.zeros(shape=(height, width, disparity), dtype=np.uint32)

right_cost_volume = np.zeros(shape=(height, width, disparity), dtype=np.uint32)

lcensus = np.zeros(shape=(height, width), dtype=np.int64)

rcensus = np.zeros(shape=(height, width), dtype=np.int64)

for d in range(0, disparity):

    rcensus[:, (x_offset + d):(width - x_offset)] = right_census_values[:, x_offset:(width - d - x_offset)]

    left_xor = np.int64(np.bitwise_xor(np.int64(left_census_values), rcensus))

    left_distance = np.zeros(shape=(height, width), dtype=np.uint32)

    while not np.all(left_xor == 0):

        tmp = left_xor - 1

        mask = left_xor != 0

        left_xor[mask] = np.bitwise_and(left_xor[mask], tmp[mask])

        left_distance[mask] = left_distance[mask] + 1

    left_cost_volume[:, :, d] = left_distance

```

```

    lcensus[:, x_offset:(width - d - x_offset)] = left_census_values[:, (x_offset +
d):(width - x_offset)]

    right_xor = np.int64(np.bitwise_xor(np.int64(right_census_values), lcensus))

    right_distance = np.zeros(shape=(height, width), dtype=np.uint32)

    while not np.all(right_xor == 0):

        tmp = right_xor - 1

        mask = right_xor != 0

        right_xor[mask] = np.bitwise_and(right_xor[mask], tmp[mask])

        right_distance[mask] = right_distance[mask] + 1

        right_cost_volume[:, :, d] = right_distance

```

```

dusk = t.time()

print('\t(done in {:.2f}s)'.format(dusk - dawn))

return left_cost_volume, right_cost_volume

```

```

def select_disparity(aggregation_volume):
    """
    last step of the sgm algorithm, corresponding to equation 14 followed by winner-takes-all approach.

    :param aggregation_volume: H x W x D x N array of matching cost for all defined directions.

    :return: disparity image.

    """
    volume = np.sum(aggregation_volume, axis=3)

```

```

disparity_map = np.argmin(volume, axis=2)

return disparity_map


def normalize(volume, parameters):
    """
    transforms values from the range (0, 64) to (0, 255).

    :param volume: n dimension array to normalize.

    :param parameters: structure containing parameters of the algorithm.

    :return: normalized array.

    """

    return 255.0 * volume / parameters.max_disparity


def get_recall(disparity, gt, args):
    """
    computes the recall of the disparity map.

    :param disparity: disparity image.

    :param gt: path to ground-truth image.

    :param args: program arguments.

    :return: rate of correct predictions.

    """

    gt = np.float32(cv2.imread(gt, cv2.IMREAD_GRAYSCALE))

    gt = np.int16(gt / 255.0 * float(args.disp))

    disparity = np.int16(np.float32(disparity) / 255.0 * float(args.disp))

    correct = np.count_nonzero(np.abs(disparity - gt) <= 3)

```

```
    return float(correct) / gt.size

def sgm_disparity(left_image_name, right_image_name, filename, parameters_,
paths_, save_images_):
    """
    main function applying the semi-global matching algorithm.

    :param left_image_name : file name image left
    :param right_image_name : file name image right
    :param parameters_ : represent all parameters used in the sgm algorithm.
    :param paths_ : represent the relation between the directions
    :param save_images_ : save
    :return: void.

    """

left_name = left_image_name
right_name = right_image_name
parameters = parameters_
paths = paths_
save_images = save_images_
output_name = '' + filename +'disparity_map.png'

dawn = t.time()

print('\nLoading images...')
left, right = load_images(left_name, right_name, parameters)
```

```

print('\nStarting cost computation...')

left_cost_volume, right_cost_volume = compute_costs(left, right, parameters,
save_images)

if save_images:

    left_disparity_map = np.uint8(normalize(np.argmin(left_cost_volume, axis=2),
parameters))

    cv2.imwrite(" + filename +'disp_map_left_cost_volume.png', left_disparity_map)

    right_disparity_map = np.uint8(normalize(np.argmin(right_cost_volume, axis=2),
parameters))

    cv2.imwrite(" + filename +'disp_map_right_cost_volume.png',
right_disparity_map)

print('\nStarting left aggregation computation...')

left_aggregation_volume = aggregate_costs(left_cost_volume, parameters, paths)

print('\nSelecting best disparities...')

left_disparity_map = np.uint8(normalize(select_disparity(left_aggregation_volume),
parameters))

if save_images:

    cv2.imwrite(" + filename +'left_disp_map_no_post_processing.png',
left_disparity_map)

print('\nApplying median filter...')

left_disparity_map = cv2.medianBlur(left_disparity_map, parameters.bsize[0])

```

```

if save_images:

    np.save(" + filename +'sgm_disparity_matrix.npy', left_disparity_map)

    cv2.imwrite(f'left_{output_name}', left_disparity_map)

dusk = t.time()

print('\nFin.')

print('\nTotal execution time = {:.2f}s'.format(dusk - dawn))

return left_disparity_map

```

Chương trình pointCloud_visualize

```

from __future__ import print_function

from open3d import *

import numpy as np

import cv2 as cv


ply_header = """ply

format ascii 1.0

element vertex %(vert_num)d

property float x

property float y

property float z

property uchar red

property uchar green

property uchar blue

end_header

```

```

"""

def write_ply(fn, verts, colors):
    verts = verts.reshape(-1, 3)
    colors = colors.reshape(-1, 3)
    verts = np.hstack([verts, colors])

    nz = (verts == np.inf).sum(1)
    q = verts[nz == 0, :]

    with open(fn, 'wb') as f:
        f.write((ply_header % dict(vert_num=len(q))).encode('utf-8'))
        np.savetxt(f, q, fmt='%.f %.f %.f %d %d %d ')

def create_pointCloud(disparity, Q, img_covert, plyFilename):
    """
    :param disparity: import disparity
    :param Q: 4x4 perspective transformation matrix
    :param img_covert: import image left/right to convert point cloud with texture color
    :param plyFilename: path contains the calibration images, import even image
    :return: point cloud file ply
    """

    points = cv.reprojectImageTo3D(disparity, Q)
    colors = cv.cvtColor(img_covert, cv.COLOR_BGR2RGB)
    # mask = disparity > disparity.min()
    mask = disparity > 70

```

```

out_points = points[mask]

out_colors = colors[mask]

out_fn = '.\\ply\\pointcloud' + plyFilename + '.ply'

write_ply(out_fn, out_points, out_colors)

print('%s saved' % out_fn)

print('Done')

def visualize_plyfile(plyFilename):

    cloud = open3d.io.read_point_cloud(plyFilename) # Read the point cloud

    open3d.visualization.draw_geometries([cloud]) # Visualize the point cloud

if __name__ == '__main__':

    filename = "4"

    disparity = np.load(" + filename + 'sgm_disparity_matrix.npy')

    Q = np.float32([[1, 0, 0, -1.57634773e+02],

                    [0, 1, 0, -2.50476748e+02],

                    [0, 0, 0, 886.73137837],

                    [0, 0, 3.43809206e-01, 0]])

    img_left = cv.imread('.\\remap\\' + filename + 'left.jpg')

    create_pointCloud(disparity, Q, img_left, filename)

    visualize_plyfile('.\\ply\\pointcloud' + filename + '.ply')

```

