Project 2: Automatic Hamburger Stacking System using ROS Robot and Deep Learning

Course: Industrial AI & Automation 2024

Name: Yunki Noh(21800226), Hyeonho Moon(21900258), Garam Jin(21900727)

Date: 2024-12-20

Youtube(Demo): https://youtu.be/UsQ8xB651xo

Youtube(Origin): https://youtu.be/99DSSQbniwo?si=2f9L64BYM88VVH6n

Instagram: https://www.instagram.com/reel/DDggm7aThcx/?utm_source=ig_web_copy_link

Python Source Link: https://github.com/JinGaram/AIAI

1. Introduction

1.1. Background

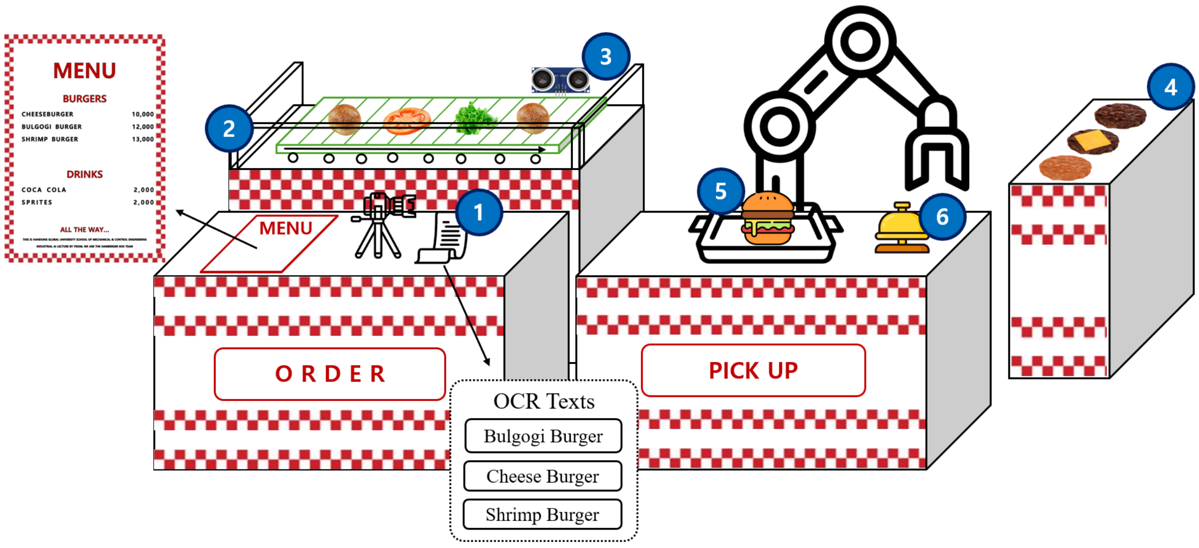


Fig 1. 3D Schematic of ROS Hamburger System

The food tech market is growing quite fast. Especially in the hamburger market, the market size has been expanded almost to about 5 trillion won. In the previous method, almost hamburger robot system was for the cooking patties of hamburger. However, if the process of stacking each ingredients of hamburger can be performed automatically, making hamburger would be performed more efficiently. In this reason, we built up overall automatic stacking system for hamburger in this project.

2. Automatic Hamburger Stacking System

In this system, three types of burger would be stacked automatically: Bulgogi Burger, Cheese Burger, and Shrimp Burger. To recognize the types of burger, we used OCR(Optimal Character Recognition) deep learning model. If someone want to order Bulgogi Burger, they need to pick down the OCR text card of 'Bulgogi Burger' under the camera 1. After 15 seconds of detecting 'Bulgogi Burger' from OCR model, Indy 10 start to pick up each ingredients (bread, bulgogi patty, Lettuce, tomato, bread) and stack up on the pick up table. After finishing stacking overall ingredients of hamburger, this robot would ring the bell.

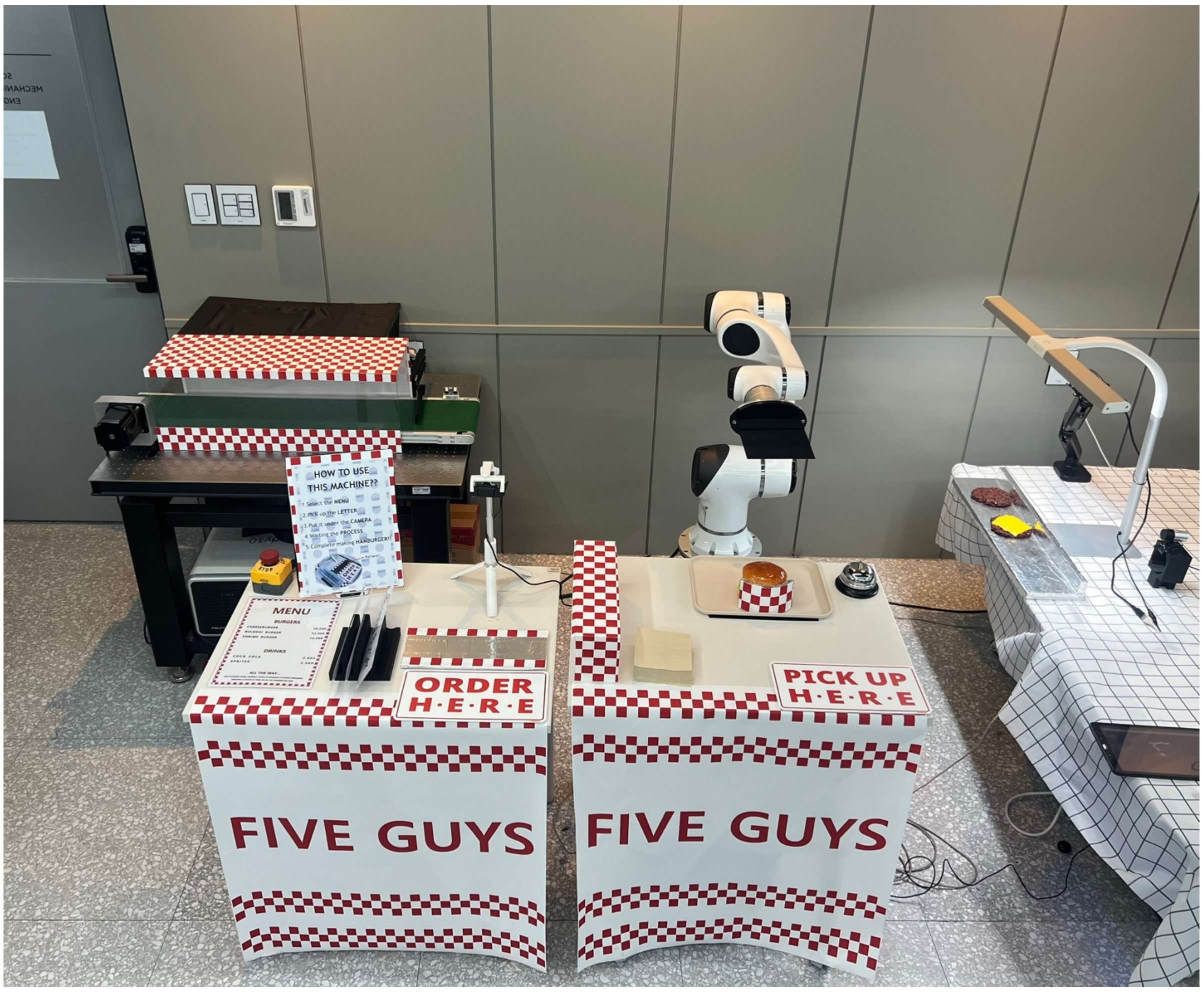


Fig 2. ROS System Set-Up Environment

2.1. Requirements for the System

- **Hardware**
  - Robot: Indy10
  - Camera1: USB camera
  - Camera2: USB camera
  - Arduino Uno R3
  - Two Stepper Motors / Two Stepper Drivers (MSD-224)
  - Ultrasonic Sensor
  - Two Power Supplies
  - Stand Light

- **Software**
  - Ubuntu 20.04
  - ROS noetic
  - Pytorch
  - OCR
  - Arduino

2.2. Build up the System

2.2.1. ROS Codes

- Terminal 1

```
roslaunch indy10_moveit_config moveit_planning_execution.launch
robot_ip:=192.168.0.9
```

To build a robot system, connection to the Indy 10 and the server is required. Both must use the same Wi-Fi network, and our group used the IP address 192.168.0.9. After connecting via the Ubuntu terminal, the robotic arm would be controlled.

- Terminal 2

```
rosrun indy_driver camera.py
```

publish: `camera/image_raw2`

This node generates image data from the camera sensor. The camera is installed above the patty for image processing.

- Terminal 3

```
rosrun indy_driver image_display.py
```

subscribe: `camera/image_raw2`

This node subscribes to the information from the camera above the patty and displays the images in a window for the user to view.

- Terminal 4

```
rosrun indy_driver image_processing.py
```

publish: `image_processing/object_info`

subscribe: `camera/image_raw2`, `ham_classifier/ham_info`, `RobotState_info`

This node can be considered the most critical node in the build system. It first subscribes to the camera data that captures information about the patty. Additionally, it subscribes to information about the Robot State. Finally, it subscribes to menu input information through the camera used for user orders. Using the data from these three subscriptions, the node publishes a single piece of information. The algorithm operates only when the Robot State is 0, ensuring that no data is processed while the robot is in motion. Furthermore, when the hamburger information obtained from the first camera matches the information obtained from the second camera, the node publishes `object_info` to the next node. If the information does not match, it does not publish any data. When retrieving information through the camera installed above the patty, image processing is performed. The node processes BGR images for three types of hamburger patties and focuses on image processing within the Region of Interest (ROI).

- Terminal 5

```
rosrun indy_driver test_ocr_video.py
```

publish: `camera/image_raw`

This node generates image data from the camera sensor. It uses a camera installed above the order station to capture user order information. Upon receiving the information, it attempts OCR detection. The OCR deep learning model utilized is provided by Google API. The system connects to Google Cloud using a JSON file for integration.

- Terminal 6

```
rosrun indy_driver image_display_ocr.py
```

subscribe: `camera/image_raw2`

This node subscribes to the information from the camera above the order station and displays the images in a window for the user to view.

- Terminal 7

```
rosrun indy_driver ham_classifier.py
```

publish: `ham_classifier/ham_info`

subscribe: `camera/image_raw`, `Robot State_info`

This node subscribes to two pieces of information: the text input data from the OCR camera and the `Robot_State_info`. The algorithm does not operate while the robot is in motion. When the robot is stationary and a predefined hamburger type is detected by the camera for more than 15 seconds, the node publishes the information to the next node.

- Terminal 8

```
rosrun indy_driver test_motion.py
```

subscribe: `ham_classifier/ham_info`, `image_processing/object_info`

The final node receives two pieces of information and executes the algorithm. This algorithm contains specific procedures for each type of hamburger. Using the information received from `Image_processing`, it calculates the coordinates of the patty based on the patty's correlation equations to determine its position. Once the hamburger-making process is complete, the algorithm terminates, and the robot returns to its standby position and stops.

- write source code for nodes in `catkin_ws/src/indy_driver/src`
    - `camera.py`
    - `image_display.py`
    - `image_processing.py`
    - `test_ocr_video.py`
    - `image_display_ocr.py`
    - `ham_classifier.py`
    - `test_motion.py`

The code for `camera.py` is as follows.

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-

import rospy
import cv2
from sensor_msgs.msg import Image               # sensor_msg 패키지로부터
Image type을 import함
from cv_bridge import CvBridge, CvBridgeError   # cv_bridge 라이브러리 :
OpenCV 이미지와 ROS 메시지 간의 변환 가능

class CameraNode2:

    def __init__(self):
        rospy.init_node('camera_node2', anonymous=True)  # 노드 이름
"camera_node"로 초기화
        self.bridge = CvBridge()                # cv_bridge 객체 생성

        # Get camera number from ROS parameter, default to 0
        camera_number = rospy.get_param('~camera_number', 0)
        rospy.loginfo(f"Camera number received: {camera_number}")

        # "camera/image_raw"라는 토픽으로 메시지를 publish할 publisher 객체
생성
        self.image_pub =
rospy.Publisher("camera/image_raw2",Image,queue_size=1)

        # self.cap = cv2.VideoCapture(camera_number)          # 카메라 연
결을 위한 VideoCapture 객체 생성
        self.cap = cv2.VideoCapture(0)
    def run(self):
        rate = rospy.Rate(30)                               # 루프 실행 주기 :
30hz
        while not rospy.is_shutdown():                      # ROS가 종료되지
않은 동안
            ret, frame = self.cap.read()                    # 카메라로부터 이미
지를 읽음
            if ret:                                         # 이미지가 정상적으
로 읽혀진 경우
                try:
                    # 읽어들인 이미지를 ROS Image 메시지로 변환하여 토픽으로
publish

 self.image_pub.publish(self.bridge.cv2_to_imgmsg(frame, "bgr8"))

                except CvBridgeError as e:
                    print(e)                                # CvBridge 변환
예외 처리
            rate.sleep()                                    # 지정된 루프 실행
주기에 따라 대기

if __name__ == '__main__':
    try:
        camera = CameraNode2()         # CameraNode 객체 생성
        camera.run()                 # run 메서드 실행
    except rospy.ROSInterruptException:
        pass
```

The code for `image_display.py` , `image_display_ocr.py` is as follows.

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-

import rospy
from sensor_msgs.msg import Image                       # sensor_msgs 패키지로
부터 Image 메시지 타입을 import
from cv_bridge import CvBridge, CvBridgeError           # cv_bridge 라이브러리
: OpenCV 이미지와 ROS 메시지 간의 변환 가능
import cv2                                              # OpenCV 라이브러리

class DisplayNode:

    def __init__(self):
        self.bridge = CvBridge()
        self.image_sub =
rospy.Subscriber("camera/image_raw",Image,self.callback)   #
camera/image_raw 토픽에서 Image 메시지 수신

    def callback(self,data):
        try:
            # 수신된 Image 메시지를 OpenCV 이미지로 변환
            cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
        except CvBridgeError as e:
            print(e)
        cv2.imshow("Camera", cv_image)  # 변환된 이미지를 "Camera"라는 이름의 윈도우에 표시
        cv2.waitKey(1)                  # 1ms 동안 키보드 입력 대기

    def run(self):
        rospy.init_node('display_node', anonymous=True) # 노드 초기화 및 이름 설정
        rospy.spin()                                    # 노드가 종료될 때까지 계속 실행

if __name__ == '__main__':
    try:
        display = DisplayNode()     # DisplayNode 클래스의 인스턴스 생성
        display.run()               # 노드 실행
    except rospy.ROSInterruptException:
        pass
```

- add message in `catkin_ws/src/indy_driver/msg`
    - `object_info.msg`

      ```
      string names
      float64 x_coords
      float64 y_coords
      float64 z_coords
      ```

    - `ham_info.msg`

      ```
      string name
      ```

- modify the file `CMakeList.txt` in `catkin_ws/src/indy_driver`

2.2.2. ROS Nodes

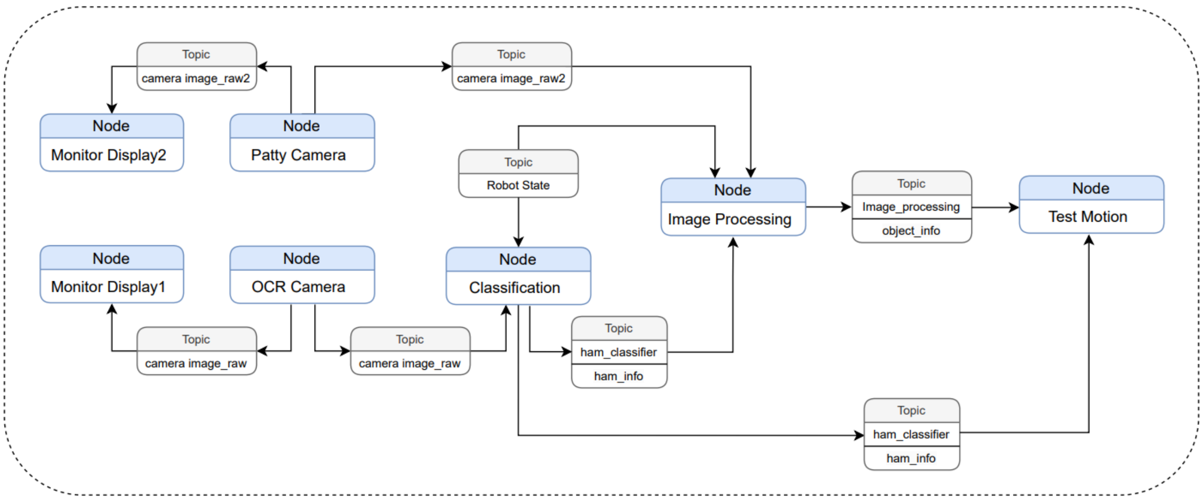


Fig 3. Diagram of ROS Process

The build system described above can be visually represented as shown in the diagram. As mentioned earlier, information is received through two camera nodes. Each camera publishes information to a display node that presents it to the user. The patty information is published to the Image Processing node, while the user’s order information is published to the Classification node.

The Image Processing node processes the images to identify the patty and determine its coordinates, which are then published to the Test Motion node. The Classification node classifies the user's order details and publishes relevant burger information to both the Image Processing and Test Motion nodes. Finally, the Test Motion node utilizes ROS to operate based on the provided information.

2.2.3. Equipment for Transferring Hamburger Ingredients

2.2.3.1 Gripper

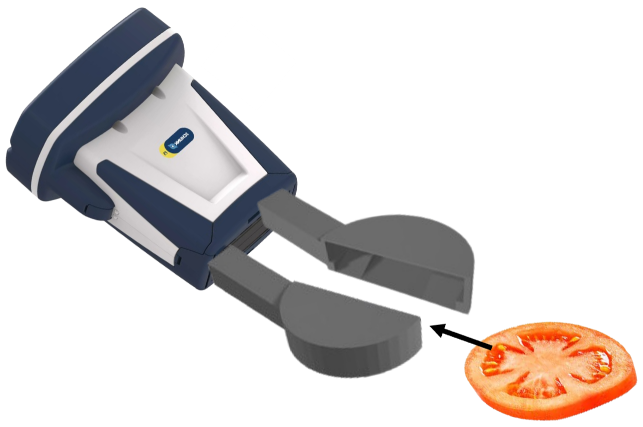


Fig 4. Gripper Design

The first proposed Burger Ingredient Transfer Model was designed in a form similar to the figure 4. This gripper was designed with a circular space and a slope to ensure that various materials could be stably positioned and easily gripped. The circular space plays a role in keeping the material centered to prevent shaking, while the slope gently guides the material to settle into an appropriate position, offering structural advantages. This design was intended to efficiently handle materials of various sizes and shapes; however, several issues were revealed during actual use.

The limited operating range of 2 cm in the motor attached to the gripper acted as a major constraint. This posed a significant problem given that the sizes of hamburger ingredients were not uniform. The hamburger materials exhibited a size difference of approximately 3 cm between the smallest and largest pieces, which made it difficult for the gripper with a limited range of motion to grip all sizes of materials. When the gripper was designed to accommodate smaller materials, it failed to grip the largest materials. Conversely, when designed for larger materials, smaller materials were not held securely or ended up falling.

These constraints necessitated a fundamental reconsideration of the gripper design. In particular, the need arose to expand the motor's operating range or introduce a flexible structure capable of adjusting to various sizes to grip materials stably.

2.2.3.2 Slider

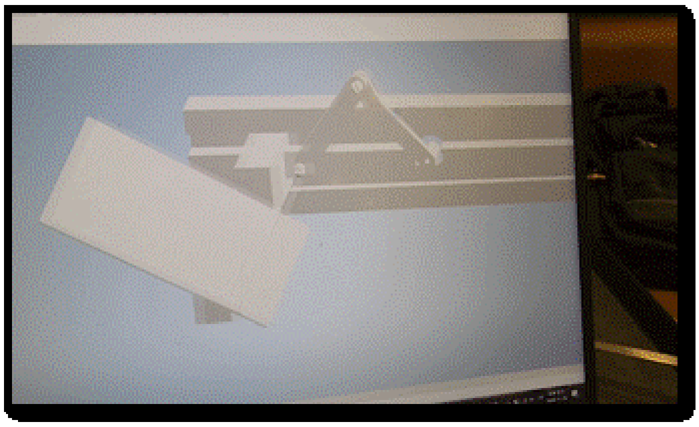


Fig 5. Slider Design

To address the first issue, the previously used gripper was removed, and a new hardware system in the form of a slider, resembling the one shown in the figure 5, was designed by combining a stepper motor and a motor driver (MSD-224). This hardware was designed to provide flexibility in handling and placing materials regardless of their size or shape. Specifically, the slider system focused on ensuring stable movement of materials and was constructed as an improved structure to overcome the limitations of the previous system.

However, during actual operation, an issue arose where the slider failed to move backward due to insufficient motor output. This led to reduced system efficiency and instability in repetitive tasks.

Thus, it was deemed more suitable to adopt a rack-and-pinion gear drive mechanism rather than the existing link-based driving system for hardware design based on the structure shown in the figure 5. The rack-and-pinion mechanism transfers driving force through the combination of a gear and a rack, resolving the problem of insufficient output and ensuring more stable operation. Furthermore, this mechanism maintains stability under load, making it effective for handling tasks involving heavy materials. Nevertheless, due to the tight project deadline, a hardware system resembling a turner, as shown in the figure 6, was designed instead.

2.2.3.3 Turner

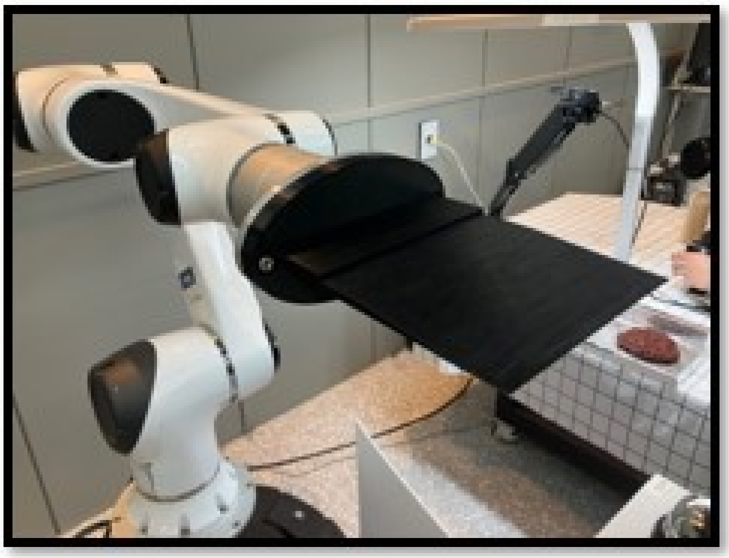
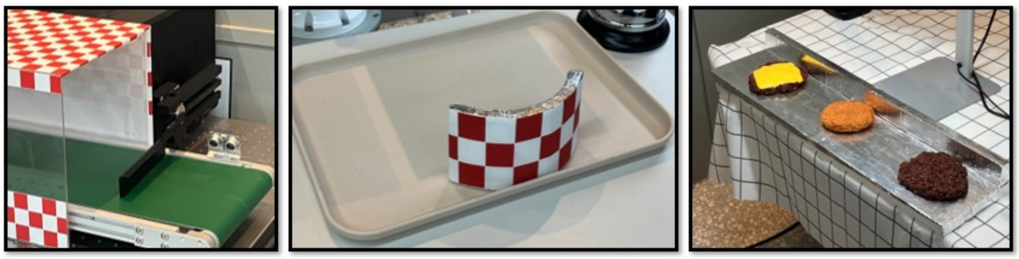


Fig 6. Turner Design

To address the issues identified in Figures 4 and 5, we newly designed a spatula-type hardware system as shown in Figure 6. This hardware was designed to efficiently transport materials, aiming to facilitate the movement and placement of materials. However, in the initial design, an issue arose where materials moved unintentionally when being transferred or placed. This highlighted the need for a guard device to securely maintain the position of the materials.

To address this, a small guard was additionally designed at the back of the grill to securely hold materials like patties when placed. This can be seen in Figures 7 (c). The guard prevents materials from sliding backward, significantly improving the stability of operations. Furthermore, to prevent materials from protruding forward during the assembly or placement of hamburgers, a hamburger barrier was added, as shown in Figure 7 (b). This barrier effectively prevented material loss during hamburger assembly.

Lastly, to sort and transport materials efficiently using a conveyor belt system, a barrier was constructed using a stepper motor and a motor driver (MSD-224). This barrier was designed to control the flow of materials and arrange them in the manner required by the operator. As a result, the level of automation in the process was greatly enhanced, creating an environment where operators could achieve the desired outcomes with minimal manual intervention.



(a) Fence on the Conveyor Belt (b) Fence on the Pick up Place (c) Fence on the Grill

Fig 7. Blocking the ingredients

2.2.4. Conveyor Belt System

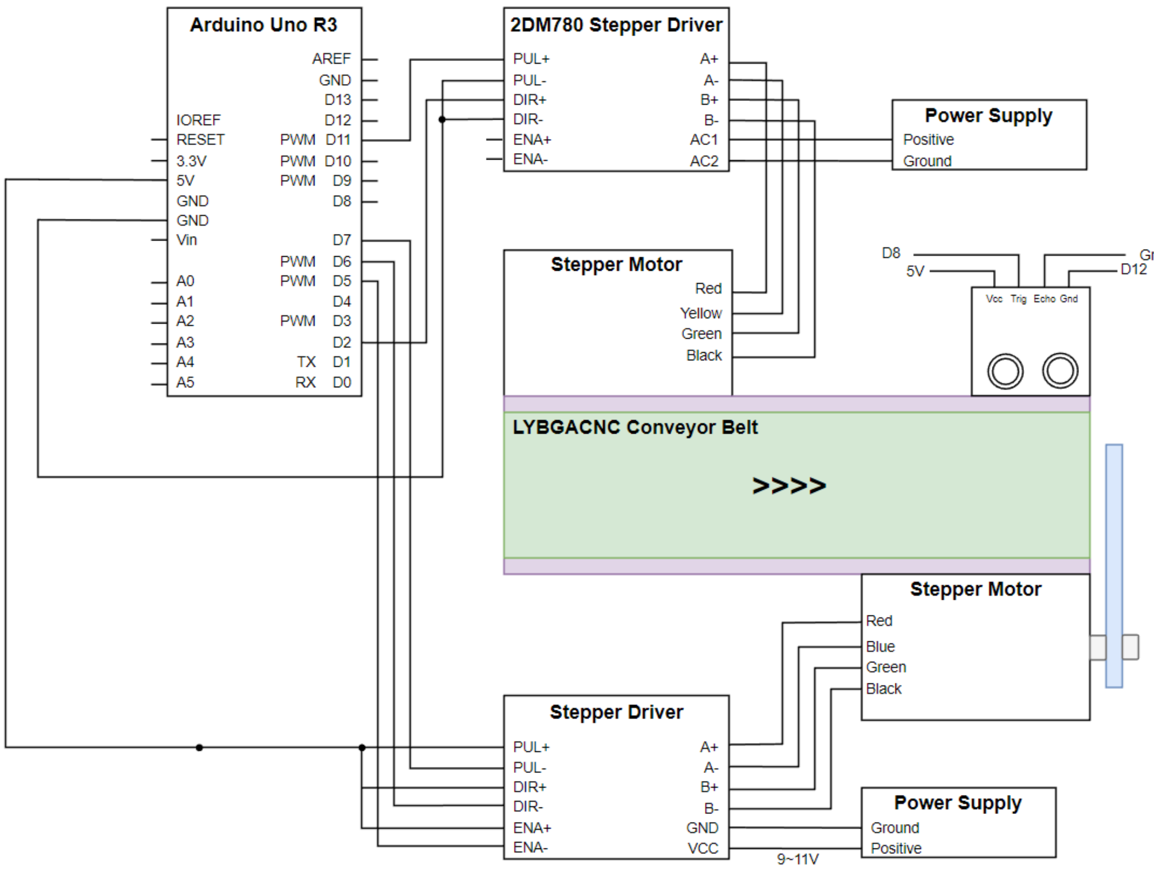


Fig 8. Design of a Conveyor Belt Using Arduino [1]

To carry each ingredient in front of the Indy 10 robot arm after this robot pick up each ingredient, we used the conveyor belt with ultrasonic. If there are not any ingredients in front of the ultrasonic sensor, the flag for running conveyor belt is turned on, and this conveyor belt will start to work. At the same time, because each ingredient need to be carried on the conveyor belt, the pence would be opened. However, if there are any ingredients in front of the ultrasonic sensor, the flat for running the conveyor belt is turned off, and this conveyor belt stops working. At this moment, to help to pick up each ingredients on the flipper, the pence would be closed. By repeating this carrying and pence system of Arduino, each ingredient can be carried in front of the robot arm automatically.

`Project2_TwoStepper.ino` : Working conveyor belt and fence with ultrasonic sensor

```
// Stepper Motor for door
int PUL = 7;  // (PUL-) of Stepper for door
int DIR = 6;  // (DIR-) of Stepper for door
int ENA = 5;  // (EN-) of Stepper for door

//Conveyor Belt
const int pwmPin = 11;  // PWM Pin of Conveyor Belt
const int dirPin = 2;   // DIrection Pin of Conveyor Belt

// UltraSonic Sensor
const int echo = 8;     // Eho of Ultrasonic
const int trig = 12;    // Trigger Pin of Ultrasonic

// Delay for Ultrasonic Sensor
```

```
unsigned long previousMillis = 0;  // For Counting Delay
const long interval = 50;  // Threshold for Delay

// Signal for Detecting Object in front of the UltraSonic Sensor
int signal = 0;

// Variables for Working door
float count1 = 0;
float count2 = 0;

// Setup for Initial Pin setting
void setup() {
    pinMode(PUL, OUTPUT);     // Stepper for door
    pinMode(DIR, OUTPUT);     // Stepper for door
    pinMode(ENA, OUTPUT);     // Stepper for door
    pinMode(dirPin, OUTPUT);  // Conveyor Belt
    pinMode(pwmPin, OUTPUT);  // Conveyor Belt
    pinMode(trig, OUTPUT);    // UltraSonic
    pinMode(echo, INPUT);     // UltraSonic
    Serial.begin(9600);       // Start Serial communication
}

void loop() {
    static float distance = 0;                // UltraSonic Distance
    unsigned long currentMillis = millis();  // Current Time

    // Delay Upload of Distance from UltraSonic Sensor
    if (currentMillis - previousMillis >= interval) { // if time is over 50ms
        previousMillis = currentMillis;  // Reset Delay Counting
        distance = measureDistance();    // Upload Distance
    }

    // Stepper for Conveyor Belt
    if (distance > 30) { // Object X
        signal = LOW; // Signal for No Object
        analogWrite(pwmPin, 127);  // Speed
        digitalWrite(dirPin, LOW); // Forward
    } else { // Object O
        signal = HIGH; // Signal for Object
        analogWrite(pwmPin, 0);  // Stop
        digitalWrite(dirPin, LOW);
    }

    // Stepper for door
    if (signal == HIGH) { // if Object O
        if (count1 == 30) { // Wating for the moment to close the door
            // Close the door
            for (int j = 0; j < 300; j++) {
                    digitalWrite(DIR, LOW);
                    digitalWrite(ENA, HIGH);
                    digitalWrite(PUL, HIGH);
                    delayMicroseconds(100);
                    digitalWrite(PUL, LOW);
                    delayMicroseconds(100);
                }
            digitalWrite(ENA, LOW);  // Deactivate motor after movement
        }

        count1 += 1; // Counting 30 times
        count2 = 0; // Reset
    }
    else if (signal == LOW) { // if Object X
        // Open
        if (count2 == 1) { // Open Only Once After Object Disappears
            // Open the door
            for (int i = 0; i < 300; i++) {
                    digitalWrite(DIR, HIGH);
                    digitalWrite(ENA, HIGH);
                    digitalWrite(PUL, HIGH);
                    delayMicroseconds(100);
                    digitalWrite(PUL, LOW);
                    delayMicroseconds(100);
                }
            digitalWrite(ENA, LOW);  // Deactivate motor after movement
        }
        count2 += 1; // Counting Only Once time
        count1 = 0; // Reset
    }
}

// Function for UltraSonic Sensor
float measureDistance() {
    digitalWrite(trig, LOW);
    delayMicroseconds(2);
    digitalWrite(trig, HIGH);
    delayMicroseconds(10);
    digitalWrite(trig, LOW);
    float cycletime = pulseIn(echo, HIGH);
    float distance = ((340.0 * cycletime) / 10000.0) / 2.0;
    return distance;
}
```

2.3. System Working

2.3.1. Execute Each Code

To execute whole system of this project, follow under sequence.

- Terminal 1

```
roslaunch indy10_moveit_config moveit_planning_execution.launch
robot_ip:=192.168.0.9
```

- Terminal 2

```
rosrun indy_driver camera.py
```

- Terminal 3

```
rosrun indy_driver image_display.py
```

- Terminal 4

```
rosrun indy_driver image_processing.py
```

- Terminal 5

```
rosrun indy_driver test_ocr_video.py
```

- Terminal 6

```
rosrun indy_driver image_display_ocr.py
```

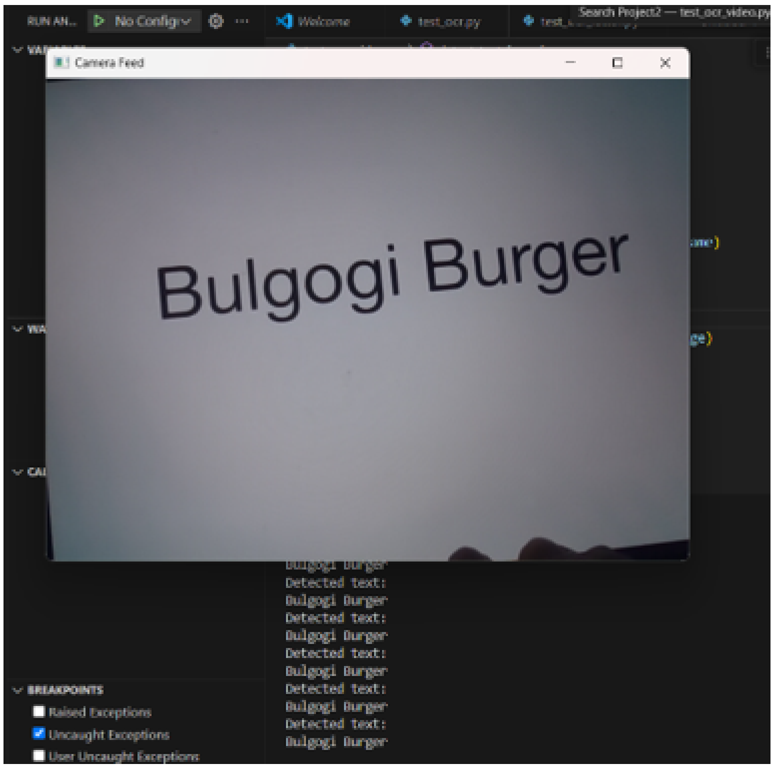
- Terminal 7

```
rosrun indy_driver ham_classifier.py
```

- Terminal 8

```
rosrun indy_driver test_motion.py
```

2.3.2. Detect the Hamburger Order using OCR Deep Learning



```
Bulgogi Burger
Detected text:
Bulgogi Burger
Detected text:
Bulgogi Burger
Detected text:
Bulgogi Burger
Detected text:
Bulgogi Burger
Detected text:
Bulgogi Burger
```

Fig 9. OCR Deep Learning System Example Picture

To start this system, we need to inform what kind of hamburger we're going to make through the camera for OCR.

The OCR (Optical Character Recognition) functionality provided by Google API is available through the Google Cloud Vision API. This API detects and extracts text from images. It automatically identifies text in images and returns the text data regardless of the language, supporting multilingual text detection, including Korean.

For convenience, a sheet listing three menu options is set up at the order station. When the sheet is placed in the designated position, the system automatically detects the text. If the text is continuously detected for 15 seconds, the next algorithm is triggered. The results are returned in JSON format, which includes the text and its coordinate information.

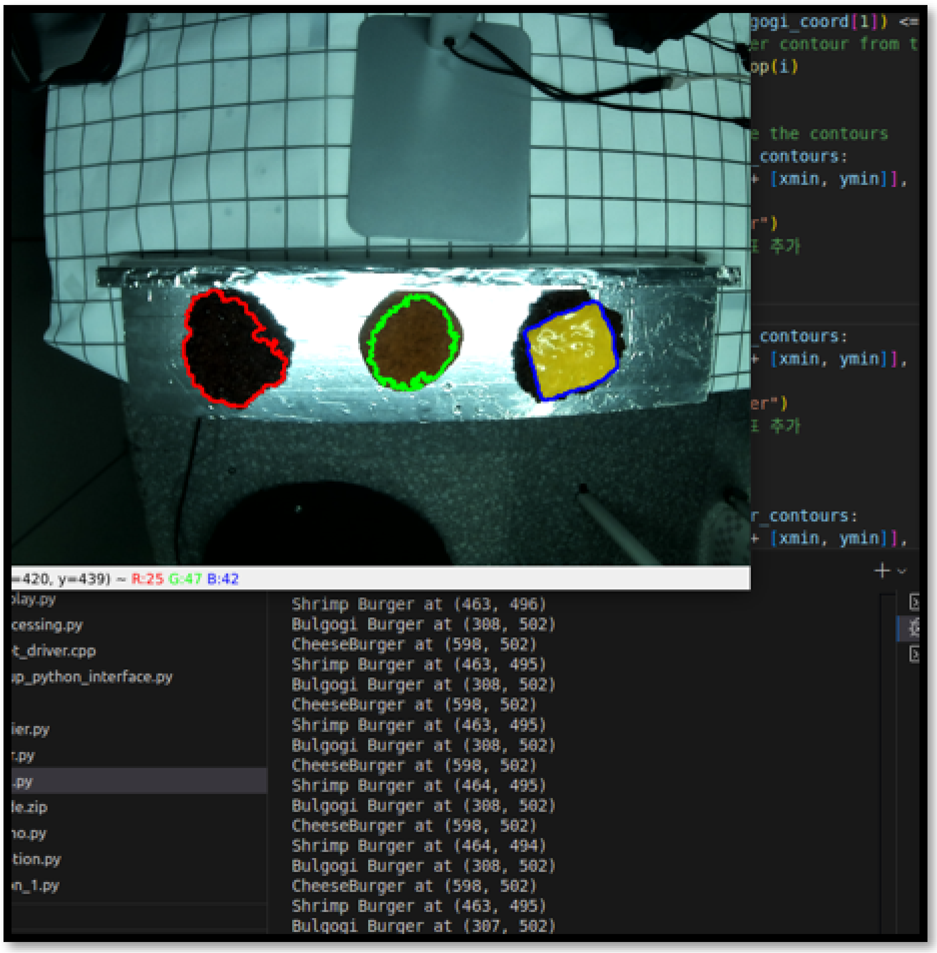
While QR codes could have been used to receive order information, the OCR technique was chosen instead. In an unmanned burger shop, printing QR codes for three items and placing them risks users mixing them up. With QR codes, users cannot know the content until they scan it with the camera. However, with text-based orders, there is no confusion for users. Therefore, the text-based OCR technique was selected. In this case, we informed the 'Bulgogi Burger' text to the OCR camera.

`test_ocr_video.py` : Detecting the text of hamburger order through OCR model

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-

import cv2
from google.cloud import vision
import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError

class cameranode:

    def __init__(self):
        rospy.init_node('camera_node', anonymous=True)  # 노드 이름 "camera_node"로 
초기화
        self.bridge = CvBridge()                # cv_bridge 객체 생성

        # Get camera number from ROS parameter, default to 0
        camera_number = rospy.get_param('~camera_number', 2)
        rospy.loginfo(f"Camera number received: {camera_number}")

        # "camera/image_raw"라는 토픽으로 메시지를 publish할 publisher 객체 생성
        self.image_pub = rospy.Publisher("camera/image_raw",Image,queue_size=1)


        # self.cap = cv2.VideoCapture(camera_number)          # 카메라 연결을 위한 
VideoCapture 객체 생성
        self.cap = cv2.VideoCapture(2)


    def main(self):
        # OpenCV를 통해 카메라 접근
        rate = rospy.Rate(30)                           # 루프 실행 주기 : 30hz
        while not rospy.is_shutdown():                  # ROS가 종료되지 않은 동안
            ret, frame = self.cap.read()                # 카메라로부터 이미지를 읽음
            if ret:                                     # 이미지가 정상적으로 읽혀진 
경우
```

```
                try:
                    # 읽어들인 이미지를 ROS Image 메시지로 변환하여 토픽으로 publish
                    #self.detect_text_from_frame(frame)
                    self.image_pub.publish(self.bridge.cv2_to_imgmsg(frame,
"bgr8"))

                except CvBridgeError as e:
                    print(e)                                # CvBridge 변환 예외 처리
              rate.sleep()                                  # 지정된 루프 실행 주기에 따
라 대기

if __name__ == "__main__":
    camera = cameranode()
    camera.main()
```

2.2.3. Detect Each Patty using Image Processing



```
Shrimp Burger at (463, 496)
Bulgogi Burger at (308, 502)
CheeseBurger at (598, 502)
Shrimp Burger at (463, 495)
Bulgogi Burger at (308, 502)
CheeseBurger at (598, 502)
Shrimp Burger at (463, 495)
Bulgogi Burger at (308, 502)
CheeseBurger at (598, 502)
Shrimp Burger at (464, 495)
Bulgogi Burger at (308, 502)
CheeseBurger at (598, 502)
Shrimp Burger at (464, 494)
Bulgogi Burger at (308, 502)
CheeseBurger at (598, 502)
Shrimp Burger at (463, 495)
Bulgogi Burger at (307, 502)
```

Fig 10. Patty Image Processing

One of the most challenging tasks in the hamburger production process was image processing. Specifically, strong reflections from the grill and the characteristics of wide-angle cameras made accurate object detection difficult. Initially, when performing image processing against a white background, converting images to grayscale to detect objects and obtain their central coordinates was relatively straightforward. Additionally, coordinate calculations with random movement along the X and Y axes were feasible.

However, improvements in image processing techniques became necessary to accurately distinguish patty types according to hamburger orders and place them on the grill. For this purpose, a process was added to classify the patties, extract their coordinates, and convert them into absolute coordinates usable in ROS.

At first, attempts were made to distinguish patties using the HSV color model, but its sensitivity to changes in lighting intensity limited its practicality. To address this, a stand providing consistent lighting was installed, but limitations still persisted. Subsequently, the BGR color model was utilized to define unique color ranges for each patty, and a detection algorithm was built based on these ranges. To differentiate patty colors, an InRange tracking bar was added to the video output window, allowing estimation of each patty's color range. Through this, the color ranges for bulgogi, shrimp, and cheese patties were defined and optimized.

However, under certain conditions, the contours of bulgogi and shrimp patties overlapped. To resolve this, a code was written to remove shrimp patty contours from bulgogi patty contours if the difference in their x-coordinates was below a certain threshold. This approach enabled more accurate contour estimation for each patty.

Based on the contours extracted through InRange processing, the coordinates of each patty were calculated, and absolute coordinates were scaled for use in ROS. This allowed estimation of the grill's starting and ending coordinates and implementation of a code capable of precisely placing the patties.

Through these processes, an image processing and control system capable of accurately distinguishing patty types and placing them in the desired locations under various order conditions was successfully completed.

`image_processing.py` : Performing image processing to detect each type of patty.

```
#!/usr/bin/env python3
# -*- coding:utf-8 -*-

import rospy
from sensor_msgs.msg import Image
from cv_bridge import CvBridge, CvBridgeError
import cv2 as cv
import numpy as np
from indy_driver.msg import object_info, robot_state, hamburger_info

class ImageProcessingNode:

    def __init__(self):
        # CvBridge
        self.bridge = CvBridge()

        # Subscrib
        self.sub_image = rospy.Subscriber("camera/image_raw2", Image,
self.callback_image)
        self.sub_robot = rospy.Subscriber("robot_state", robot_state,
self.chk_robot)
        self.sub_ham = rospy.Subscriber("ham_classifier/ham_info",
hamburger_info, self.callback_ham)

        self.bAction = False  # 로봇 상태에 따른 동작 여부 판단
        self.hamburger_name = None  # 햄버거 정보 저장
        # Publisher
        self.pub_object_info = rospy.Publisher("image_processing/object_info",
object_info, queue_size=10)
        self.msg_object_info = object_info()  # 퍼블리셔로 보낼 메시지 객체 생성

        # 배열 저장
        self.object_names = []
        self.x_coords = []
        self.y_coords = []
        self.z_coords = []

    def callback_ham(self, data):
        """햄버거 정보 콜백 함수"""
        # 수신한 햄버거 정보를 저장
        self.hamburger_name = data.name
        rospy.loginfo(f"Received hamburger info: {self.hamburger_name}")

        # 로봇이 햄버거를 찾을 준비가 되면, 동작 시작
        self.bAction = True

    def callback_image(self, data):
        """이미지 처리 콜백 함수"""
        if self.bAction:  # 로봇이 햄버거를 찾을 준비가 되었을 때만 동작
            try:
                frame = self.bridge.imgmsg_to_cv2(data, "bgr8")
                frame = cv.resize(frame, (640, 480))  # 이미지 크기 조정

                roi_points = [(110, 205), (615, 205), (115, 345), (604, 335)]
                roi_points = np.array(roi_points, dtype=np.int32).reshape((-1, 1,
2))

                xmin = min([point[0][0] for point in roi_points])
                ymin = min([point[0][1] for point in roi_points])
                xmax = max([point[0][0] for point in roi_points])
                ymax = max([point[0][1] for point in roi_points])
                roi = frame[ymin:ymax, xmin:xmax]

                # HSV 색상 범위 정의 (Yellow, Brown, Dark Brown)
                h_min_y, h_max_y = 0, 34  # Yellow Hue 범위
                s_min_y, s_max_y = 0, 255
                v_min_y, v_max_y = 100, 240

                h_min_b, h_max_b = 0, 20  # Brown Hue 범위
                s_min_b, s_max_b = 0, 255
                v_min_b, v_max_b = 0, 76

                h_min_bb, h_max_bb = 0, 68  # Dark Brown Hue 범위
                s_min_bb, s_max_bb = 0, 255
                v_min_bb, v_max_bb = 0, 40

                #HSV변환
                hsv_frame = cv.cvtColor(roi, cv.COLOR_BGR2HSV)

                #마스크
                mask_yellow = cv.inRange(hsv_frame, (h_min_y, s_min_y, v_min_y),
(h_max_y, s_max_y, v_max_y))
                mask_brown = cv.inRange(hsv_frame, (h_min_b, s_min_b, v_min_b),
(h_max_b, s_max_b, v_max_b))
                mask_dark_brown = cv.inRange(hsv_frame, (h_min_bb, s_min_bb,
v_min_bb), (h_max_bb, s_max_bb, v_max_bb))

                contours_yellow = []
                contours_brown = []
                contours_dark_brown = []

                # 컨투어
                contours_yellow, _ = cv.findContours(mask_yellow,
cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)
                contours_brown, _ = cv.findContours(mask_brown, cv.RETR_EXTERNAL,
cv.CHAIN_APPROX_SIMPLE)
                contours_dark_brown, _ = cv.findContours(mask_dark_brown,
cv.RETR_EXTERNAL, cv.CHAIN_APPROX_SIMPLE)

                # Calculate centroids and classify burgers
                shrimp_burger_contours = []
                bulgogi_burger_contours = []
                cheese_burger_contours = []


                # 각 햄버거에 컨투어
                for contour in contours_yellow:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            cheese_burger_contours.append((contour, cx + xmin, cy
+ ymin))
                            #print(f"CheeseBurger at ({cx + xmin}, {cy + ymin})")

                for contour in contours_brown:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            shrimp_burger_contours.append((contour, cx + xmin, cy
+ ymin))
                            #print(f"Shrimp Burger at ({cx + xmin}, {cy +
ymin})")

                for contour in contours_dark_brown:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            bulgogi_burger_contours.append((contour, cx + xmin,
cy + ymin))
                            #print(f"Bulgogi Burger at ({cx + xmin}, {cy +
ymin})")

                for i, shrimp_coord in enumerate(shrimp_burger_contours[:]):  #
Copy list for safe iteration
                    for bulgogi_coord in bulgogi_burger_contours:
                        if abs(shrimp_coord[1] - bulgogi_coord[1]) <= 5:  #
Compare x-coordinates
                            # Remove the shrimp burger contour from the list
                            shrimp_burger_contours.pop(i)
                            break


                for contour, cx, cy in cheese_burger_contours:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        cv.drawContours(frame, [contour +[xmin, ymin]], -1, (255,
0, 0), 2)

                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            print(f"acheese_burger_contours ({cx})")
                            if self.hamburger_name == "Cheese Burger":
                                self.object_names.append("Cheese Burger")
                                self.x_coords.append(cx)
                                self.y_coords.append(cy)
                                self.z_coords.append(0)

                for contour, cx, cy in shrimp_burger_contours:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        cv.drawContours(frame, [contour + [xmin, ymin]], -1, (0,
255, 0), 2)

                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            print(f"shrimp_burger_contours ({cx})")
                            if self.hamburger_name == "Shrimp Burger":
                                self.object_names.append("Shrimp Burger")
                                self.x_coords.append(cx)
                                self.y_coords.append(cy)
                                self.z_coords.append(0)

                for contour, cx, cy in bulgogi_burger_contours:
                    area = cv.contourArea(contour)
                    if 3800 < area < 8000:  # Filtering based on area
                        cv.drawContours(frame, [contour + [xmin, ymin] ], -1, (0,
0, 255), 2)

                        M = cv.moments(contour)
                        if M["m00"] != 0:
                            cx = int(M["m10"] / M["m00"])
                            cy = int(M["m01"] / M["m00"])
                            print(f"bulgogi_burger_contours ({cx})")
                            if self.hamburger_name == "Bulgogi Burger":
                                self.object_names.append("Bulgogi Burger")
                                self.x_coords.append(cx)
                                self.y_coords.append(cy)
                                self.z_coords.append(0)

                if self.object_names:
                    self.msg_object_info.names = self.object_names[-1]  # 가장 최
근의 햄버거 이름
                    self.msg_object_info.x_coords = self.x_coords[-1]  # 가장 최근
의 x 좌표
                    self.msg_object_info.y_coords = self.y_coords[-1]  # 가장 최근
의 y 좌표
                    self.msg_object_info.z_coords = self.z_coords[-1]  # 가장 최근
의 z 좌표
                    self.pub_object_info.publish(self.msg_object_info)
                    rospy.loginfo(f"Published {self.hamburger_name} at
({self.msg_object_info.x_coords})")

                cv.imshow("Frame", frame)
                cv.imshow("roi", roi)
                cv.waitKey(1)

            except CvBridgeError as e:
                rospy.logerr(f"CV Bridge Error: {e}")

    def chk_robot(self, msg):
    #"""로봇 상태 처리 콜백 함수"""
        if msg.move == 1:
            self.bAction = False
        else:
            self.bAction = True  # 로봇이 멈추면 bAction을 False로 설정

if __name__ == '__main__':
    rospy.init_node('image_processing_node')
    image_processing_node = ImageProcessingNode()
    rospy.spin()
```

2.3.4. Repeat to pick up and stack each ingredient

The overall sequence of stacking hamburger process would be unified except for the patties depending on the type of hamburger. The process would repeat the pick up and stack up process with each ingredient.

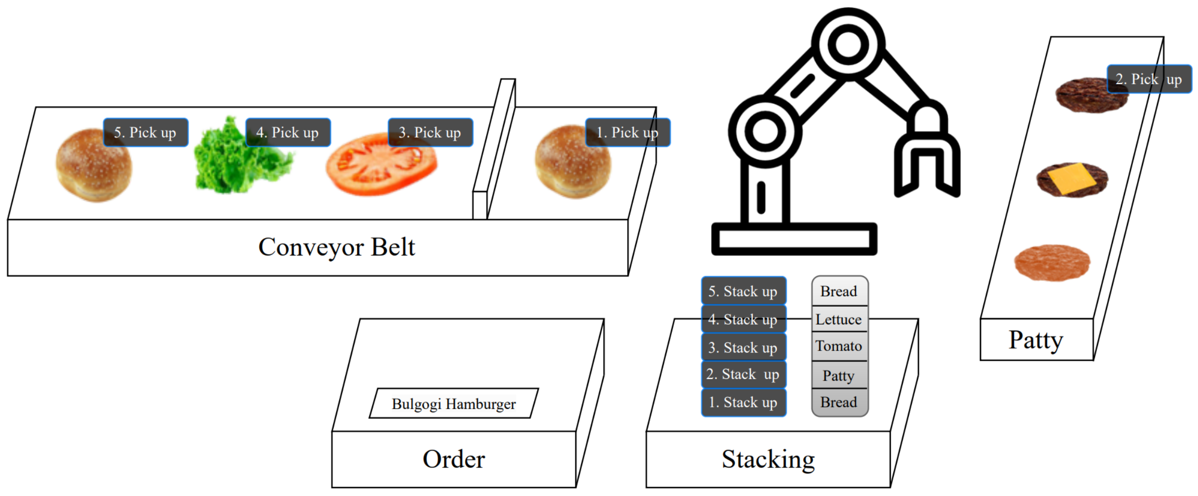


Fig 11. Overall Process of Repeating to Pick up and Stack Each Ingredient

In the case of Bulgogi Hamburger, this robot would pick up the bread and place it on the stacking position. After this, this robot would turn to the right to pick up patty. In this case, the bulgogi patty would be picked up and would be placed on bread. Then this robot would pick up and place each ingredients (tomato, lettuce, and bread) repeatedly.

`ham_classifier.py` : Delivering hamburger types based on OCR results

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-

import rospy
from sensor_msgs.msg import Image    # sensor_msgs 패키지로부터 Image 메시지 타입을
import
from cv_bridge import CvBridge, CvBridgeError      # cv_bridge 라이브러리 : OpenCV
이미지와 ROS 메시지 간의 변환 가능
import cv2                           # OpenCV 라이브러리
import numpy as np
from google.cloud import vision


from indy_driver.msg import robot_state, hamburger_info


class HamClassifierNode():
    def __init__(self):
        rospy.init_node('ham_classifier', anonymous=True) # 노드 초기화 및 이름 설정
        self.bridge = CvBridge()


        self.bAction = False
        # Subscriber
        self.sub_msg = rospy.Subscriber("camera/image_raw", Image, self.action)
 # camera/image_raw 토픽에서 Image 메시지 수신
        self.sub_robot = rospy.Subscriber("robot_state", robot_state,
self.chk_robot)  # camera/image_raw 토픽에서 Image 메시지 수신

        # Publisher
        self.pub_ham_info = rospy.Publisher('ham_classifier/ham_info',
hamburger_info, queue_size=10)
```

```
        # Message Control Variables
        self.bSend = False
        self.bAction = False
        self.ham_info = hamburger_info()
        self.thresh_cnt = 15
        self.chk_cnt = 0
        self.ham_id_prev = 0
        self.ham_id_curr = 0


    def chk_robot(self,robot_state):
        if robot_state.move == 1:
            self.bAction = False
        else:
            self.bAction = True

    def action(self,data):
        if self.bAction:
            try:

                cv_image = self.bridge.imgmsg_to_cv2(data, "bgr8")
                #Google Vision API 클라이언트 생성
                client = vision.ImageAnnotatorClient()

                # 프레임을 바이너리 데이터로 변환
                _, encoded_image = cv2.imencode('.jpg', cv_image)
                content = encoded_image.tobytes()

                # Vision API에 이미지 전송 및 텍스트 인식
                image = vision.Image(content=content)
                response = client.text_detection(image=image)
                label = response.text_annotations

                if label:
                    label = label[0].description
                else:
                    label = "No text detection"

                if label == "Cheese Burger":
                    self.ham_id_curr = 1
                elif label == "Bulgogi Burger":
                    self.ham_id_curr = 2
                elif label == "Shrimp Burger":
                    self.ham_id_curr = 3
                else:
                    self.ham_id_curr = 0

                if (self.ham_id_curr == self.ham_id_prev) and self.ham_id_curr !=
0:
                    self.chk_cnt += 1
                else:
                    self.chk_cnt = 0

                if self.chk_cnt >= self.thresh_cnt:
                    self.chk_cnt = 0
```

```
                        self.bSend = True

                    if self.bSend:
                        self.ham_info.name = label
                        self.pub_ham_info.publish(self.ham_info)
                        self.bSend = False

                    self.ham_id_prev = self.ham_id_curr

                    rospy.loginfo("%s, %d, %d",label, self.ham_id_curr, self.chk_cnt)


                    # cv2.imshow("Camera", cv_image)  # 변환된 이미지를 "Camera"라는 이
름의 윈도우에 표시
                    # cv2.waitKey(1)                  # 1ms 동안 키보드 입력 대기

            except Exception as e:
                print(e)

    def run(self):
        rospy.spin()                                        # 노드가 종료될 때까지 계속
실행


if __name__ == '__main__':
    try:
        ham_classifier = HamClassifierNode()       # CameraNode 객체 생성
        ham_classifier.run()                  # run 메서드 실행
    except rospy.ROSInterruptException:
        pass
```

`test_motion.py` : Working Inday 10 robot depending on the type of hamburger

```
#!/usr/bin/env python3
#-*- coding:utf-8 -*-

from move_group_python_interface import MoveGroupPythonInterface
from math import tau
import rospy

# import serial
# import time

import sys
import moveit_commander
import moveit_msgs
import geometry_msgs

import tf
import numpy as np
from math import pi
from indy_driver.msg import hamburger_info, object_info
```

```
DEG2RAD = pi/180
RAD2DEG = 180/pi




stack_info =      { "cv_top"   :      { "Bulgogi Burger": np.array([-90, -25,
115, 0, 0, 0])* DEG2RAD,
                                        "Cheese Burger"  : np.array([-90, -25,
115, 0, 0, 0])* DEG2RAD,
                                        "Shrimp Burger" : np.array([-90, -25,
115, 0, 0, 0])* DEG2RAD},
                    "cv_bottom":      { "Bulgogi Burger": np.array([-90, -8, 130,
0, -30, 0])* DEG2RAD,
                                        "Cheese Burger"  : np.array([-90, -8,
130, 0, -30, 0])* DEG2RAD,
                                        "Shrimp Burger" : np.array([-90, -8, 130,
0, -30, 0])* DEG2RAD},

                    "plate_top"   :   { "Bulgogi Burger": np.array([0, -25, 115,
0, 0, 0])* DEG2RAD,
                                        "Cheese Burger"  : np.array([0, -25, 115,
0, 0, 0])* DEG2RAD,
                                        "Shrimp Burger" : np.array([0, -25, 115,
0, 0, 0])* DEG2RAD},
                    "plate_angle":   { "Bulgogi Burger" : np.array([0, 0, 0, 0,
20, 0])* DEG2RAD,
                                        "Cheese Burger"  : np.array([0, 0, 0, 0,
20, 0])* DEG2RAD,
                                        "Shrimp Burger" : np.array([0, 0, 0, 0,
20, 0])* DEG2RAD},

                    "patty_top"   :   { "Bulgogi Burger": np.array([90, -25, 115,
0, 0, 0])* DEG2RAD,
                                        "Cheese Burger"  : np.array([90, -25,
115, 0, 0, 0])* DEG2RAD,
                                        "Shrimp Burger" : np.array([90, -25, 115,
0, 0, 0])* DEG2RAD},

                    "quantity":       { "Bulgogi Burger" : 1,
                                        "Cheese Burger"   : 1,
                                        "Shrimp Burger"  : 1},


                    "CV_front"    :    {    "rel_xyz"         : [0.13, 0, 0]
,
                                            "rel_rpy"          : [0.0, 0.0, 0.0]
     },
                    "CV_back"     :    {    "rel_xyz"         : [-0.13, 0, 0]
,
                                            "rel_rpy"          : [0.0, 0.0, 0.0]
     },

                    #patty_down"    : {     "rel_xyz"          :  [x, y, -0.48]
                    "patty_down"     : {     "down"              : np.array([90, 
22.46, 142.4, 0, -76.8, 0])* DEG2RAD},
                    "patty_md_up"    : {    "rel_xyz"           : [0, 0, 0.05]
  ,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
     },

                    "patty_md_down"  : {    "rel_xyz"           : [0, 0, -0.05]
,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
     },

                    "patty_front"    : {    "rel_xyz"           : [-0.12, 0, 0]
  ,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
     },
                    "patty_back"     : {    "rel_xyz"           : [0.12, 0, 0]
,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
     },

                    "plate_bottom":   {  "bottom1":   np.array([0, -21.7, 129.7, 
0, -18, 0])* DEG2RAD   ,
                                         "bottom2":   np.array([0, -22.35, 128.4, 
0, -16, 0])* DEG2RAD  ,
                                         "bottom3":   np.array([0, -23, 127.1, 0, 
-14.1, 0])* DEG2RAD   ,
                                         "bottom4":   np.array([0, -23.5, 125.7, 
0, -12.2, 0])* DEG2RAD ,
                                         "bottom5":   np.array([0, -23.9, 124.3, 
0, -10.4, 0])* DEG2RAD },



                    "plate_front" :   {     "rel_xyz"           : [0, 0.12, 0]
,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
      },
                    "plate_back"   :   {     "rel_xyz"           : [0, -0.12, 0]
,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
      },
                    "plate_front_lettuce":{ "rel_xyz"           : [0, 0.125, 0]
,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
      },
                    "plate_back_lettuce" :{ "rel_xyz"           : [0, -0.125, 0]
   ,
                                            "rel_rpy"           : [0.0, 0.0, 0.0]
      },

                    "bell"          :{ "up":     np.array([15, -3.15, 126.3, 0, 
-35.2, 0])* DEG2RAD   ,
                                       "down":   np.array([15, -2.64, 127.7, 0, 
-35.2, 0])* DEG2RAD  ,      }
                    }
```

```python
class HamFeederNode():
    def __init__(self):
        # Subscriber 설정
        self.sub_ham_class = rospy.Subscriber("ham_classifier/ham_info",
hamburger_info, self.stack_ham)
        self.sub_object_info = rospy.Subscriber("image_processing/object_info",
object_info, self.stack_object)

        # Robot 초기화
        self.indy10 = MoveGroupPythonInterface(real=True)
        iniplace = np.array([0, -25, 115, 0, 0, 0]) * DEG2RAD
        self.indy10.go_to_joint_abs(iniplace)

        # 저장 변수 초기화
        self.current_ham_info = None
        self.current_object_info = None

    def stack_ham(self, msg):
        # hamburger_info 메시지를 받은 경우
        self.current_ham_info = msg
        # object_info도 받았으면 stack 처리
        if self.current_object_info:
            self.process_stack()

    def stack_object(self, msg):
        # object_info 메시지를 받은 경우
        self.current_object_info = msg
        # hamburger_info도 받았으면 stack 처리
        if self.current_ham_info:
            self.process_stack()

    def process_stack(self):
        # 현재의 hamburger_info와 object_info를 처리
        hamburger_info = self.current_ham_info
        object_info = self.current_object_info

        # object_info를 이용한 x, y 좌표
        x = object_info.x_coords
        y = object_info.y_coords


        # bread (그릇 상단 -> 컨베이어 상단 -> 컨베이어 하단 -> 재료 앞으로 -> 재료 뒤로
-> 컨베이어 하단 -> 컨베이어 상단 -> 그릇 상단 -> 그릇 하단1 -> 그릇 앞으로 -> 그릇 각도 -
> 그릇 뒤로))
        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])

        self.indy10.go_to_pose_rel(stack_info["CV_front"]['rel_xyz'],
stack_info["CV_front"]['rel_rpy'])

        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])

        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])
        # lettuce (그릇 상단 -> 컨베이어 상단 -> 컨베이어 하단 -> 재료 앞으로 -> 재료 뒤로 -> 컨베이어 하단 -> 컨베이어 상단 -> 그릇 상단 -> 그릇 하단2 -> 그릇 앞으로(양상추만) -> 그릇 각도 -> 그릇 뒤로(양상추만)))
        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])


        self.indy10.go_to_pose_rel(stack_info["CV_front"]['rel_xyz'], stack_info["CV_front"]['rel_rpy'])
        # self.indy10.go_to_pose_rel(stack_info["CV_back"]['rel_xyz'], stack_info["CV_back"]['rel_rpy'])

        #self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])

        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])


        # tomato (그릇 상단 -> 컨베이어 상단 -> 컨베이어 하단 -> 재료 앞으로 -> 재료 뒤로 -> 컨베이어 하단 -> 컨베이어 상단 -> 그릇 상단 -> 그릇 하단4 -> 그릇 앞으로 -> 그릇 각도 -> 그릇 뒤로))
        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])

        self.indy10.go_to_pose_rel(stack_info["CV_front"]['rel_xyz'], stack_info["CV_front"]['rel_rpy'])


        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])

        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])


        # bread (그릇 상단 -> 컨베이어 상단 -> 컨베이어 하단 -> 재료 앞으로 -> 재료 뒤로 -> 컨베이어 하단 -> 컨베이어 상단 -> 그릇 상단 -> 그릇 하단5 -> 그릇 앞으로 -> 그릇 각도 -> 그릇 뒤로))
        self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])

        self.indy10.go_to_pose_rel(stack_info["CV_front"]['rel_xyz'], stack_info["CV_front"]['rel_rpy'])
        # self.indy10.go_to_pose_rel(stack_info["CV_back"]['rel_xyz'], stack_info["CV_back"]['rel_rpy'])



        #self.indy10.go_to_joint_abs(stack_info["cv_bottom"][hamburger_info.name])
        self.indy10.go_to_joint_abs(stack_info["cv_top"][hamburger_info.name])
```

```
            self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])

            self.indy10.go_to_joint_abs(stack_info["bell"]["up"])
            self.indy10.go_to_joint_abs(stack_info["bell"]["down"])


            self.indy10.go_to_joint_abs(stack_info["plate_top"][hamburger_info.name])

            self.current_ham_info = None
            self.current_object_info = None

        def run(self):
            rospy.spin()                                    # 노드가 종료될 때까지 계속
실행

if __name__ == '__main__':
    try:
        Ham_goooo = HamFeederNode()
        Ham_goooo.run()                # run 메서드 실행
    except rospy.ROSInterruptException:
        pass
```

3. Results and Discussion

3.1. Results

We conducted a total of 15 trials, with 5 attempts for each type of patty. Successfully calculating the coordinates and locating each patty was achieved in all 15 attempts (15/15). The ultrasonic sensor successfully stopped the materials at the appropriate positions on the conveyor belt 55 times out of 60 detections (55/60). The camera accurately detected the user's order and published the information in all 15 attempts (15/15). However, during 15 trials, there were 5 failures due to issues such as robot deviation, conveyor belt errors, and stacking failures on the final tray. Therefore, the success rate for completing hamburgers was 11 out of 15 trials (11/15). The calculated accuracy for each of the four processes is as follows: 100%, 92%, 100%, and 80%. Detailed results can be found in the table below.

Table 1. Performance of the ROS Process

	Process	Accuracy	Precision	Recall	F-1 Score
1	Patty Detection	100	100	100	100
2	Picking Materials(Lettuce, Bread, Tomato)	91.7	91.7	91.7	91.7
3	OCR Camera Detection	100	100	100	100
4	Burger Making Complete	73.3	73.3	73.3	73.3

3.2. Discussion

We set many devices to build an automatic hamburger stacking system. However, the results of our system were affected by various working environment conditions, such as unstable image processing, the hard materials of the ingredients, and the limitations of the gripper.

The final hamburger completion rate of the designed system was 73.3%, which is insufficient for practical use in the industry. Most importantly, the issue of device deviation must be addressed. Additionally, it is necessary to enhance the accuracy and detail of each process to eliminate potential variables. Although there are no overall obstruction factors, the system remains unstable. To develop a stable system for economic benefits, we must evaluate the overall feasibility of its development.

With improvements, we believe this system could achieve better results. Below are the conclusions for each part:

- Gripper: Due to the limited range of the gripper, we decided to use a flipper instead of a gripper.
- Conveyor System: The conveyor system with ultrasonic sensors was successfully built, but it is unstable due to the diverse shapes of the ingredients.
- OCR: The OCR detection system [Publish, Subscribe] was implemented successfully.
- Detect Each Patty: We could detect the types of ingredients using information from the contours. However, this system is also unstable due to its sensitivity to lighting conditions.
- Build-up Node / Publish / Subscribe: The system between nodes and topics functions successfully.

In summary, while the system works successfully overall, there are challenges in specific parts. Addressing and improving these difficulties in each part is expected to enhance the system's completeness.

4. References

[1] Sanghyeon-K. (n.d.). *Automatic Classification & Pick-and-Place Demo Tutorial*. GitHub. Retrieved November 20, 2024, from https://github.com/ykkimhgu/MIP2022-robot-sorting-indy10-demo/blob/main/Automatic%20Classification%20%26%20Pick-and-Place/Automatic%20Classification%20%26%20Pick-and-Place%20Demo%20Tutorial.md