Shoebot : automated shoes organizing system

@ Team member : Tae-Gyun Yang, Min-Woong Han, Jun-Young Choi

@ Date : Spring semester, 2023

@ Part : Automation part in Industrial AI & Automation class

@ Project #2

@ Instructor : prof. Young-Keun Kim

---

1. Introduction

The objective of this project is to develop an automated system using a robotic arm in the ubuntu 20.04 environment to place and retrieve shoes from a cabinet. The first goal is to identify appropriate motion commands for the robot and perform suitable path planning based on the given situation. The second goal is to utilize a deep learning model trained on a custom dataset to recognize shoes and generate corresponding robotic arm movements. This report provides detailed information on the appropriate format of commands for the robot, as well as the flow of the system for executing this project.

2. Requirements

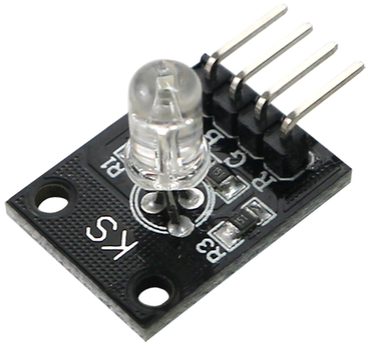
2.1. Software

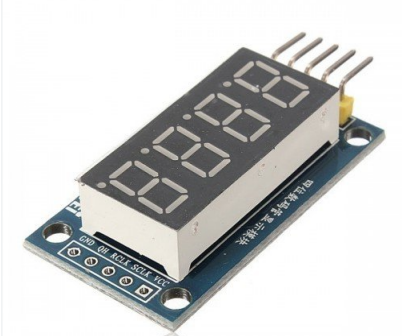
- OS : ubuntu 20.04 version
- Environment setting related to deep learning
  - GPU : NVIDIA GEOFORCE RTX 3070
  - cuda 11.3 version
  - cudnn 8.2.1 version
  - pytorch 1.21.1 + cu113 (GPU version)
  - All detailed installation guide of `Cuda & Pytorch` is specified in another md file `yolov5.md`.
- Arduino IDE
- Python for robot manipulating, deep learning based object detection

2.2. Hardware

- Microprocessor board :
  - Arduino uno x 1
- Camera sensor : Microsoft LifeCam studio (EA : 1)

- Arduino sensor : RGB led (EA : 1), 4-bit LED Digital Tube module (EA : 1)



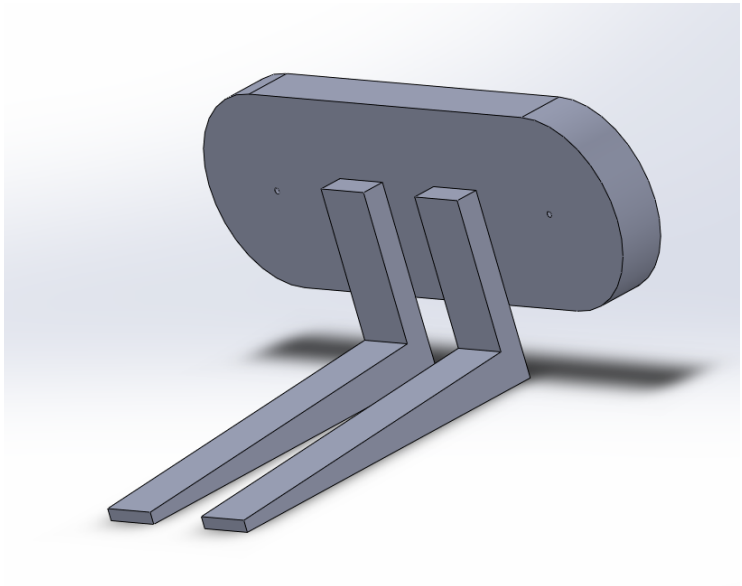


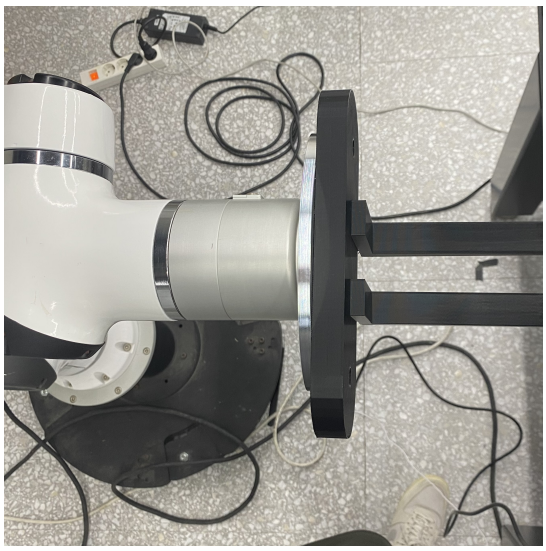
- robot arm : Indy 10

    - Specifications of Indy 10 robot
        - 6 degree of freedom
        - 10 [kg] payload
        - 175 deg
        - joint 1,2 : 60 deg/s, joint 3,4,5,6 : 90 deg/s
        - Maximum reach : 1000mm

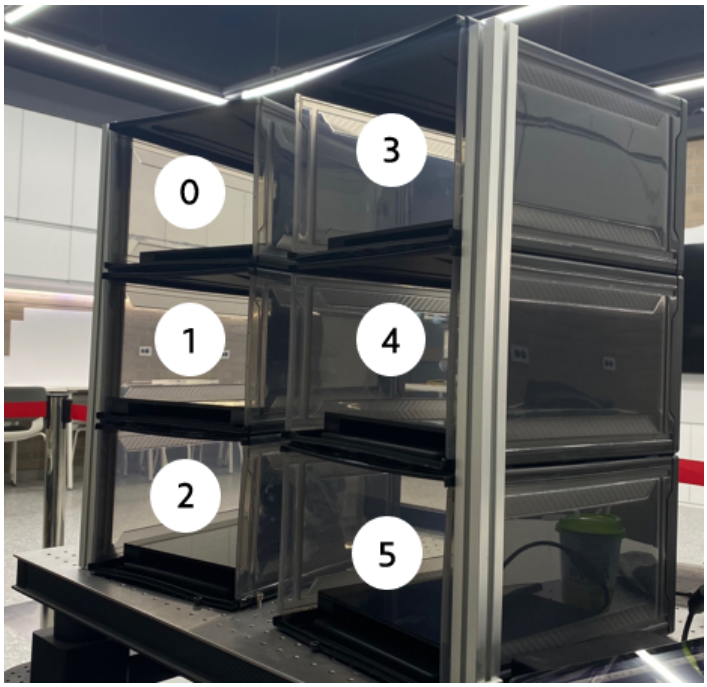
- End effect bracket combining part

  A part needs to be designed to be attached to the bracket provided with the default in the end effector. The corresponding part has been 3D printed, and here are the design details and the actual form of the attached part.





- Shoes cabinet (EA : 1)



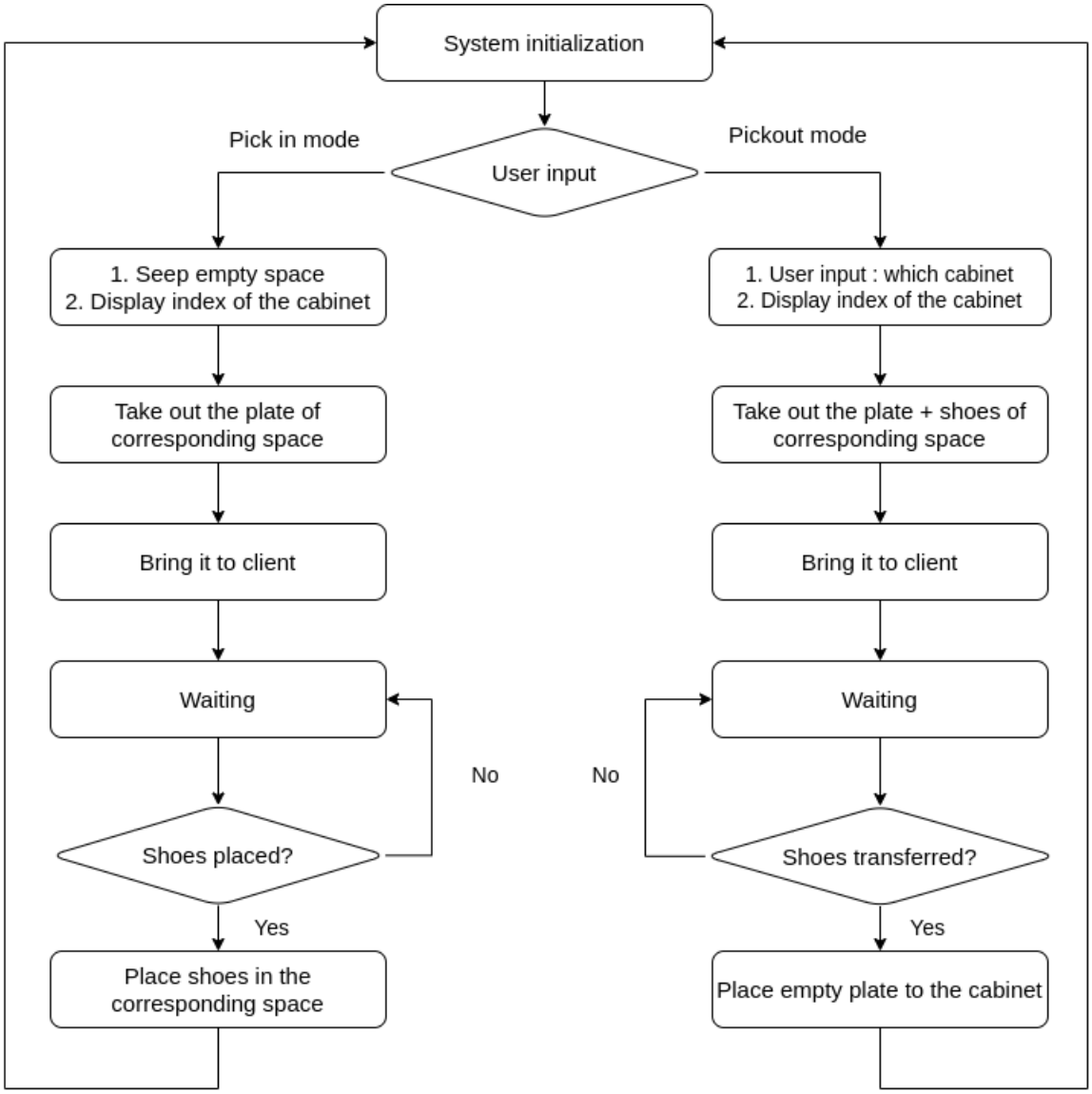
3. Process

3.1. Process structure

There are two main processes involved in this project. The first process is the robotic arm control process, which receives user input and performs the corresponding arm movements. The second process is the camera process, which utilizes a deep learning model trained on a custom dataset to determine the presence of shoes on the plate. The key aspect is the establishment of robust communication between the two processes at the appropriate timing, and the rqt_graph illustrating this communication will be shown later.

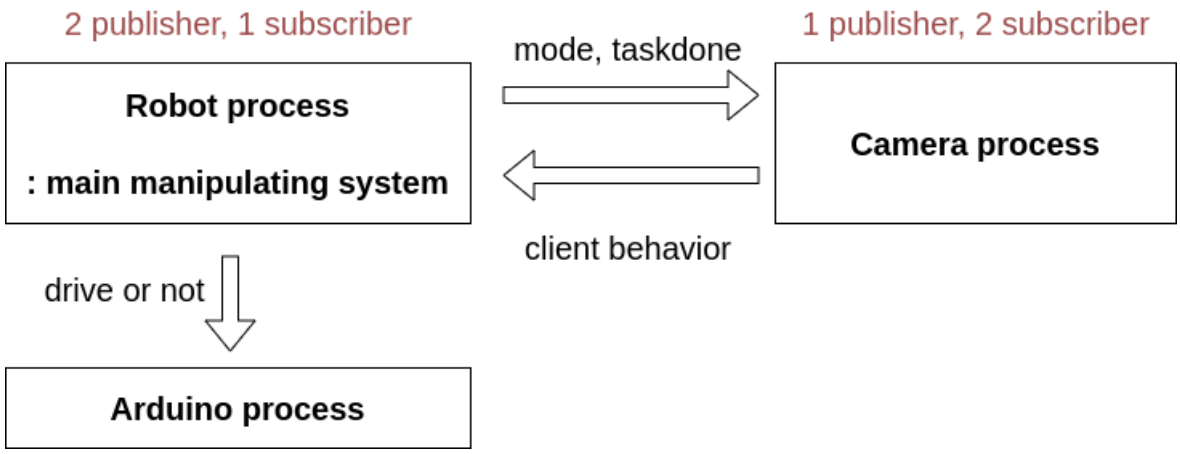
- Robot manipulating process : main process
- Camera process : deep learning utilization

3.2. System flow chart



3.3. System process

Description of the implementation components (detailed description of above flow chart)and communication structure of the system is as follows.



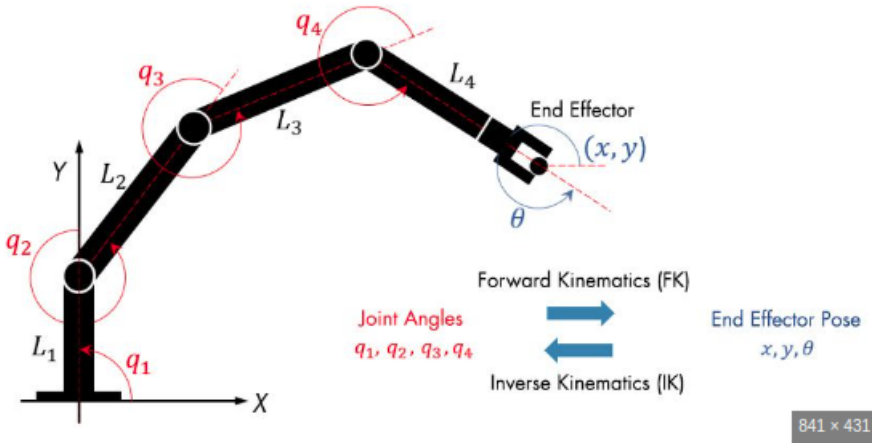
- **Implementation components**

  1. The system receives user input to determine the desired action
  - input "1" : indicates the intention to place shoes
    - Set 'modeSelection' flag with 1 & Send that flag to camera process with publisher 'taskPub'
    - Seek for empty space
    - Take out the empty plate in the corresponding space & Display index with 4-bit digital tube module
    - Robot manipulating
    - Bring the empty plate to the client
    - Set 'workdoneFlag' flag with 1 if the manipulation is completed & Send that flag to camera process with publishr 'modePub'
    - Wait until the client properly place shoes on the plate (Wait for the data receiving with subscriber)
    - Once the shoes have been placed, the system receives the information from the camera process to determine the appropriate location for inserting the shoes into the designated compartment.

  - input "2" : indicates the intention to retrieve shoes
    - Set 'modeSelection' flag with 2 & Send that flag to camera process with publisher 'taskPub'
    - The system receives user input to determine which compartment to retrieve the shoes from.
    - Display index of selected cabinet with 4-bit digital tube module
    - The system checks if the shoes are present in the specified compartment. If the shoes are not found, the system waits for further instructions regarding another compartment. (exception handling)

- After retrieving the plate with the shoes placed on it, the system proceeds to remove it from the designated place. The plate is then delivered to the client or user as requested.
- Set 'workdoneFlag' flag with 1 if the manipulation is completed & Send that flag to camera process with publishr 'modePub'
- Wait until the client properly removed shoes on the plate (Wait for the data receiving with subscriber)
- Once the user has taken the shoes, the system receives information from the camera process to confirm the absence of shoes on the plate. It then proceeds to place the empty plate back in its original position, effectively resetting the system.

3.4. Robot manipulation



There are three types of commands for robot motion: absolute coordinate commands (x, y, z, roll, pitch, yaw), relative coordinate commands (x, y, z, roll, pitch, yaw), and joint commands (commands for each joint in radians).

To achieve optimal robot motion, all types of commands were tested and simulated. In this project, precise and stable motion in a static environment is essential. Absolute & relative coordinate system has infinite numerical solution to reach to the target point . Accordingly, even when reaching the desired points, unintended paths can be taken, leading to unstable motion. To address this, joint commands to plan stable trajectories were used for path planning.

Joint commands for tasks such as picking up plates from cabinets and placing them back were stored. For each path planning, I collected the respective joint command pairs and defined a function to generate robot motion based on these commands. A function that was defined in this project takes a 2D list that contains the paths for various tasks and generates the corresponding joint commands for the robot.

- path planning

```
""" cabinet 0 planning [PICK IN] : SIMULATION COMPLETE """

caninet_0_pickin_0 = [0.08, -8.58, 86.15, 3.89, 9.36, -3.95]
caninet_0_pickin_1 = [0.04, 0.82, 77.10, 3.89, 9.35, -3.95]
caninet_0_pickin_2 = [0.04, 23.53, 50.65, 3.89, 11.11, -3.95]
caninet_0_pickin_3 = [0.04, 22.60, 50.77, 3.89, 11.12, -3.95]
caninet_0_pickin_4 = [0.03, -23.40, 97.17, 3.89, 11.14, -3.95]

cabinet_0_in_planning = [caninet_0_pickin_0, caninet_0_pickin_1,
caninet_0_pickin_2, caninet_0_pickin_3, caninet_0_pickin_4]
```

- robot manipulation

```
def jointPlanning(planning) :

    rate = rospy.Rate(10)
    indy10_interface = MoveGroupPythonInterface()

    for action in planning :

        indy10_interface.go_to_joint_state(jointSet(action))
        time.sleep(0.2)

# Application in main statement
jointPlanning(cabinet_0_in_planning)
```

3.5. Micro controller : Arduino application

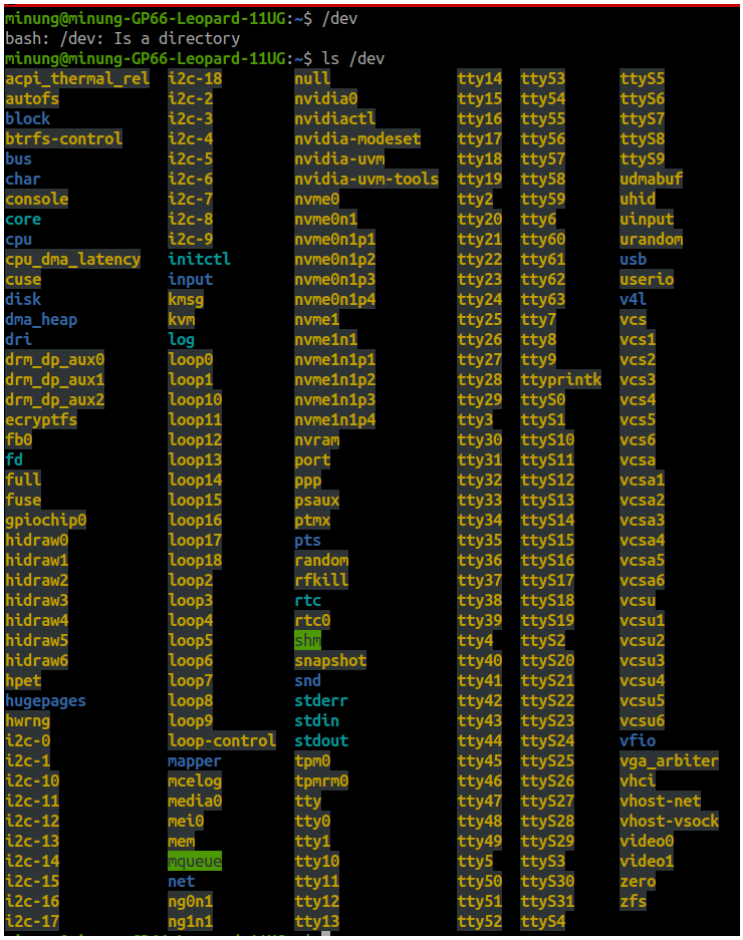
Downloaded software : Arduino IDE

download link : Click here to download

The Arduino board was utilized to provide visual effects during robot operation using an RGB sensor. It emits a red light to indicate a warning during robot motion and switches to a green light when the robot completes its operation and enters a waiting state. Additionally, a 4-bit LED digital module was utilized to visually display the assigned numbers indicating which cabinet the shoes should be placed in. This was implemented to provide a visual indication of the cabinet assignment for easy reference during the shoe placement process. Since the Arduino is connected to the PC via a USB connector, it can communicate with the main process through serial communication as long as the port number is properly configured. By leveraging this capability, relevant information can be passed to the Arduino process, such as signaling the completion of robot motion by sending an appropriate flag. This enables the Arduino to perform the necessary actions based on the received information.

When the Arduino USB cable is connected to the PC, a new port is assigned. To check which port the Arduino has been allocated, client can use the following command in terminal. Typically, the port name is `ttyACM0`.

```
ls /dev
```



In Windows environment, when checking the port number for Arduino, client needs to identify the assigned port in the Device Manager and enter the corresponding number. Unlike the `ttyACM0` format in Linux, Windows uses a different format to represent the serial ports.

The difference between the two environments can be summarized as follows :

- Windows

```
import serial

ser = serial.Serial('COM5', 9600)
```

- Ubuntu

```
import serial

ser = serial.Serial('/dev/ttyACM0', 9600)
```

In arduino process, two informations about robot manipulating is transferred to main process.

- Whether the robot is currently running
- Assigned cabinet number

To transmit both values simultaneously with serial communication, in Python, the two values are concatenated together, encoded, and then transmitted to the Arduino process. In the Arduino process, the received data is interpreted, and the two values are separated (parsing process). Based on the interpreted flags, the appropriate sensor actions are executed. The detailed code used in this project is specified in `appendix`.

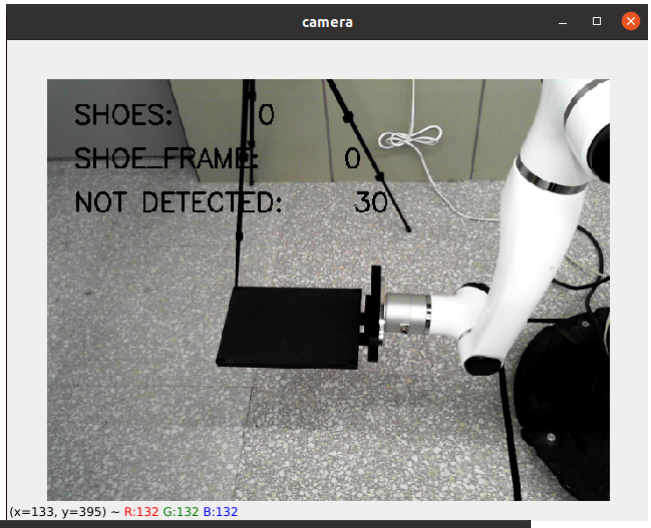
3.5. Deep learning model generation

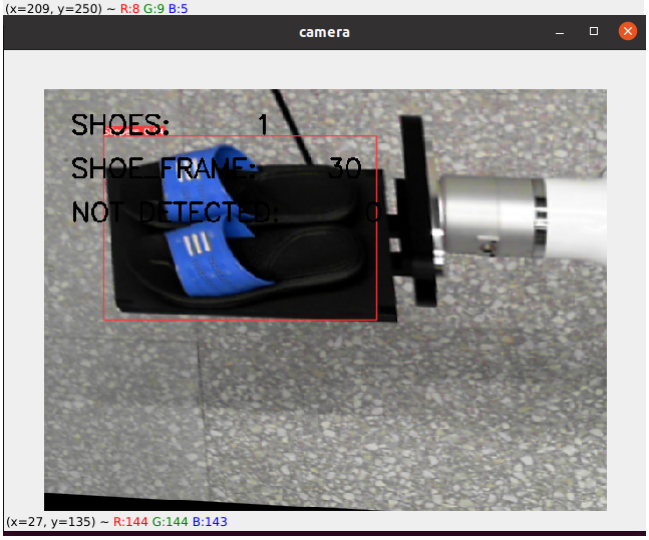
- model : YOLO V5
- number of applied images used for training set : 147 images with using 'roboflow'.

Since the recognition of objects (shoes) in a fixed and static environment was the focus, it was determined that there was no need to train on a large number of shoe images. Using Roboflow, distortions such as cropping were applied to the original images to generate additional images for preventing overfitting. A total of 147 images were used for training, and the system demonstrated appropriate recognition capability in each experiment.

3.6. Inverse perspective mapping

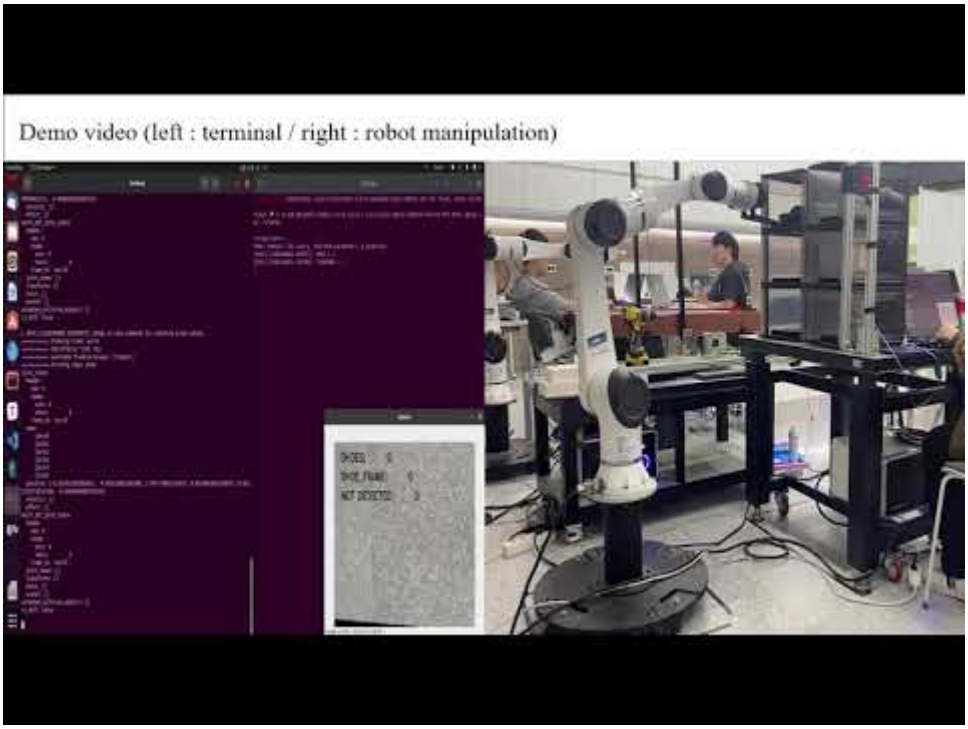
Taking into account the camera's installation angle and field of view, it is necessary to prevent incorrect recognition of objects in areas other than the region of interest. Setting a region of interest (ROI) can be one approach, but considering that the training images primarily captured shoes from a top-down angle, applying a different technique such as inverse perspective mapping was deemed necessary. OpenCV provides functions for this purpose, and inverse perspective mapping is commonly used to obtain a bird's eye view. By applying this technique, the camera's perspective was transformed. Using the model generated through `3.5. Deep learning model generation`, recognition was conducted with the following results.





4. Result (Demo video)

Video link : Click here



5. Multi process shell script management

To execute each process separately, you would need to open individual terminals, set the appropriate paths, enter "source devel/setup.bash" command, and execute each process separately. This can be cumbersome. To address this, in Windows, batch files are commonly used to manage the execution of multiple files, allowing them to be launched together. In Linux, the equivalent of a batch file is a `bash shell script`, which has a .sh extension. Through a bash shell script, execution of multiple files can be managed at once. Below is a detailed explanation of how the overall file execution is managed using a bash shell script.

5.1. shell script generation

First, a batch shell script file to manage multiple files at once must be created. To do this, enter the following command in a terminal window:

```
* filename : runSimulation

touch runSimulation.sh
chmod +x runSimulation.sh
nano runSimulation.sh

* generalized version

touch [filename].sh
chmod +x [filename].sh
nano [filename].sh
```

5.2. script description

If the commands provided above is entered sequentially, a new .sh file will open below the terminal. The desired files can be executed and managed at all at once in that location. The code represents opening the simulation file in terminal 1 and running `projectRobot.py` in terminal 2. When pressed Ctrl + X and then Y, the .sh file will be saved.

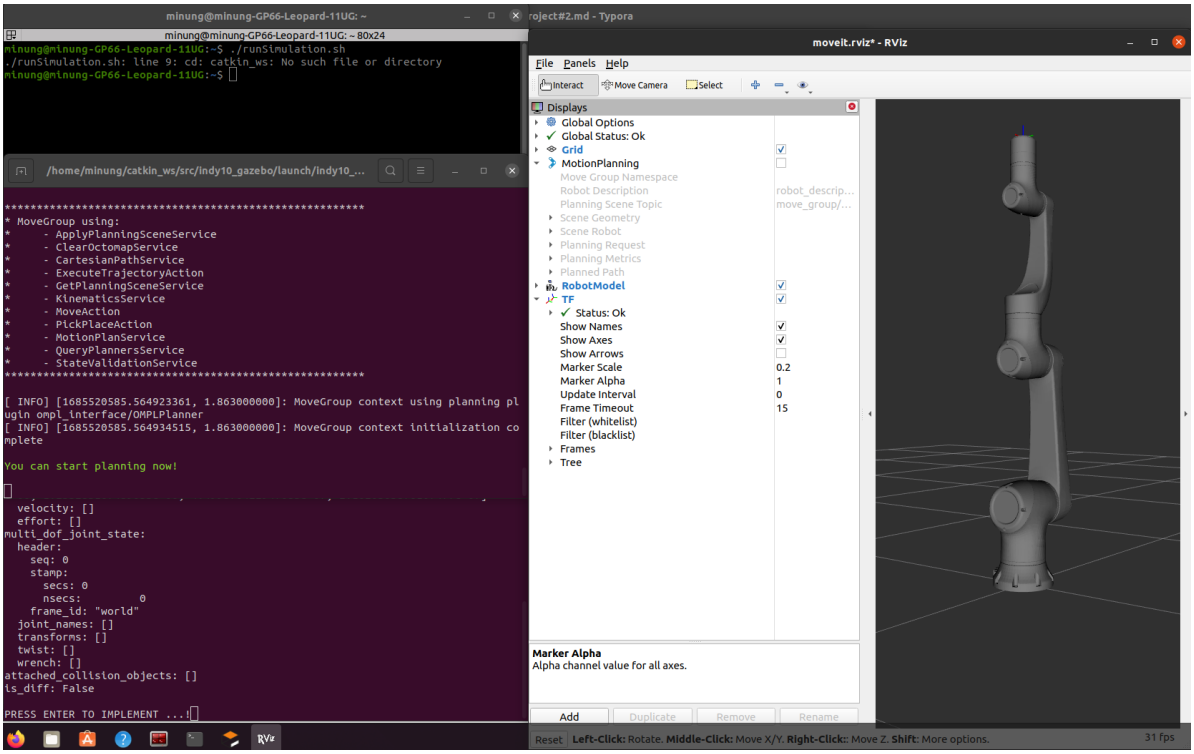
```
#!/bin/bash

# For terminal 1
cd catkin_ws
source devel/setup.bash
gnome-terminal -- /bin/sh -c 'roslaunch indy10_gazebo
indy10_moveit_gazebo.launch; exec bash'

# For terminal 2
cd catkin_ws
source devel/setup.bash
gnome-terminal -- /bin/sh -c 'rosrun indy_driver projectRobot.py; exec
bash'
```

Once the file is saved, .sh extension file can be executed using the following command. This allows user to manage and execute the files sequentially all at once. As a result, the simulation file is executed like below. Users can efficiently run and manage the desired files based on this process with all above processes.

```
./runSimulation.sh
```



6. Discussion and analysis

In robot motion control, there are three types of commands: absolute coordinates, relative coordinates, and joint commands. When using absolute or relative coordinate motion, if a command significantly deviates from the current state, there can be infinitely many solutions to reach the target position, leading to unexpected robot motions. By using information such as fraction of completion, the estimated time required for the motion can be evaluated, and if it exceeds a threshold, the intended command can be reissued. However, due to the project's requirement for fast execution of predefined motions, all motions were composed using joint commands. As a result, highly accurate motion was achieved, successfully realizing the intended motions.

7. Appendix

- Robot main code

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

from projectFunctions import *
from projectjointDef import *
from std_msgs.msg import Int32

class robotProcess:

    def __init__(self):

        self.taskdonePub      = rospy.Publisher('taskfinish', Int32,
queue_size = 10)
        self.modePub          = rospy.Publisher('modeSelect', Int32,
queue_size = 10)
        self.taskdoneSub      = rospy.Subscriber('taskFlag', Int32,
self.callback)

        self.indy10_interface = MoveGroupPythonInterface()
        self.rate             = rospy.Rate(10)

        self.msg_taskFlag     = Int32()
        self.msg_modeFlag     = Int32()

        self.workdoneFlag     = 0
        self.modeSelection    = 0
        self.camdoneFlag      = 0
```

```
        self.cabinetSeq       = 0
        self.cabinetIdx       = [0, 0, 0, 0, 0, 0]

        self.pickinPlan       = []
        self.placePlan        = []
        self.pickoutAvail     = []
        self.pickoutIdx       = None


    def main(self):

        input("PRESS ENTER TO INITIALIZE ROBOT STATE ...!\n\n")

        jointPlanning([initial_pos])

        print("INITIALIZATION COMPLETED ...!\n\n")

        try:

            while not rospy.is_shutdown() :

                self.getmodeInput()

                if(self.modeSelection == 1) :

                    self.in_exceptionHandling()

                    self.findemptySpacePick()

                    self.cabinetState()

                    self.tasksendtoCam()

                    # recieve task done flag from cam process
                    self.waitforClient()

                    self.placeShoes()

                    self.initialization()


                elif(self.modeSelection == 2) :

                    self.out_exceptionHandling()

                    self.findAvailableSpace()

                    self.selectCabinet()

                    self.robotActuation()
```

```
                    self.tasksendtoCam()

                    self.waitforClient()

                    self.placePlate()

                    self.initialization()


        except rospy.ROSInterruptException:
            return

        except KeyboardInterrupt:
            return



    def getmodeInput(self):

        printMode()

        while True :

            self.modeSelection = input("\nPLEASE INPUT MODE WHAT YOU WANT
...!\n\n")

            if self.modeSelection.isdigit():

                self.modeSelection = int(self.modeSelection)

                if self.modeSelection in [1, 2]:
                    break

                else:
                    print("INVALID INPUT ! PLEASE ENTER EITHER 1 OR 2.")

        self.msg_modeFlag.data = self.modeSelection

        # Publish flag to cam process
        self.modePub.publish(self.msg_modeFlag)


    def findemptySpacePick(self) :

        for Idx, cabinet in enumerate(self.cabinetIdx) :

            if cabinet == 0 :

                # # Robot manipulating part
                if   (Idx == 0) : self.pickinPlan = cabinet_0_in_planning
                elif (Idx == 1) : self.pickinPlan = cabinet_1_in_planning
```

```
                elif (Idx == 2) : self.pickinPlan = cabinet_2_in_planning
                elif (Idx == 3) : self.pickinPlan = cabinet_3_in_planning
                elif (Idx == 4) : self.pickinPlan = cabinet_4_in_planning
                elif (Idx == 5) : self.pickinPlan = cabinet_5_in_planning

                self.cabinetIdx[Idx] = 1

                break

        jointPlanning(self.pickinPlan)

        jointPlanning([goto_client])

        print(f"CABINET NUMBER '{Idx}' IS ASSIGNED ...!\n")

        self.cabinetSeq    = Idx
        self.modeSelection = 0


    def cabinetState(self) :

        print("-------------------------   CABINET STATE  ----------------
------------")
        print(f"|\t\tCABINET 0 : {self.cabinetIdx[0]}      |       CABINET
3 : {self.cabinetIdx[3]}\t\t|")
        print(f"|\t\tCABINET 1 : {self.cabinetIdx[1]}      |       CABINET
4 : {self.cabinetIdx[4]}\t\t|")
        print(f"|\t\tCABINET 2 : {self.cabinetIdx[2]}      |       CABINET
5 : {self.cabinetIdx[5]}\t\t|")
        print("-----------------------------------------------------------
------------")


    def in_exceptionHandling(self) :

        if all(space == 1 for space in self.cabinetIdx):

            print("ALL CABINETS IS FILLED...! ONLY PICKING OUT COMMAND IS
AVAILABLE...!")

            self.modeSelection = input("\nPLEASE INPUT MODE WHAT YOU WANT
...!\n\n")


    def tasksendtoCam(self) :

        self.workdoneFlag = 1

        self.msg_taskFlag.data = self.workdoneFlag

        self.taskdonePub.publish(self.msg_taskFlag)
```

```
    def initialization(self) :

        self.modeSelection = 0
        self.workdoneFlag  = 0
        self.camdoneFlag   = 0
        self.cabinetSeq    = 0
        self.pickoutIdx    = 0
        self.pickinPlan    = []
        self.placePlan     = []

        jointPlanning([initial_pos])


    def callback(self, data):

        self.camdoneFlag = data.data

        rospy.loginfo(f"RECEIVED TASK DONE FLAG FROM CAM =
{self.camdoneFlag}")


    def waitforClient(self) :

        while(self.camdoneFlag == 0) :

            rospy.sleep(2)
            print("WAITING FOR CLIENT...!")


    def placeShoes(self) :

        if   (self.cabinetSeq == 0) : self.placePlan =
cabinet_0_out_planning
        elif (self.cabinetSeq == 1) : self.placePlan =
cabinet_1_out_planning
        elif (self.cabinetSeq == 2) : self.placePlan =
cabinet_2_out_planning
        elif (self.cabinetSeq == 3) : self.placePlan =
cabinet_3_out_planning
        elif (self.cabinetSeq == 4) : self.placePlan =
cabinet_4_out_planning
        elif (self.cabinetSeq == 5) : self.placePlan =
cabinet_5_out_planning

        jointPlanning(self.placePlan)


    def out_exceptionHandling(self) :
```

```
        if all(space == 0 for space in self.cabinetIdx):

            print("THERE'S NO CABINET WITH SHOES...! ONLY PICK IN COMMAND
IS AVAILABLE ...!")

            self.modeSelection = input("\nPLEASE INPUT MODE WHAT YOU WANT
...!\n\n")


    def findAvailableSpace(self) :

        print("-----------------------------------------------------\n")


        for Idx, cabinet in enumerate(self.cabinetIdx):

            if cabinet == 1 :

                print(f"\nCANINET '{Idx}' IS AVAILABLE TO PICK OUT...!")

                self.pickoutAvail.append(Idx)


    def selectCabinet(self) :

        self.cabinetState()

        while True :

            self.pickoutIdx = input("PLEASE INPUT CABINET # NUMBER THAT YOU
WANT TO PICK OUT...!")

            if self.pickoutIdx.isdigit() :

                self.pickoutIdx = int(self.pickoutIdx)

                if self.pickoutIdx in self.pickoutAvail :

                    self.cabinetIdx[self.pickoutIdx] = 0

                    break

                else :
                    print("INVALID INPUT...! THE CABINET NUMBER YOU ENTERED
DOES NOT HAVE ANY SHOES...!")

            else :
                print("INVALID INPUT...! PLEASE ENTER A VALID CABINET
NUMBER...!")
```

```
    def robotActuation(self) :

        if  (self.pickoutIdx == 0)   :   self.pickinPlan =
cabinet_0_in_planning
        elif(self.pickoutIdx == 1)   :   self.pickinPlan =
cabinet_1_in_planning
        elif(self.pickoutIdx == 2)   :   self.pickinPlan =
cabinet_2_in_planning
        elif(self.pickoutIdx == 3)   :   self.pickinPlan =
cabinet_3_in_planning
        elif(self.pickoutIdx == 4)   :   self.pickinPlan =
cabinet_4_in_planning
        elif(self.pickoutIdx == 5)   :   self.pickinPlan =
cabinet_5_in_planning

        jointPlanning(self.pickinPlan)

        print("PLATE IS TRANSFERRING TO CLIENT ...!")

        jointPlanning([goto_client])


    def placePlate(self) :

        if   (self.pickoutIdx == 0)   :   self.placePlan =
cabinet_0_out_planning
        elif (self.pickoutIdx == 1)   :   self.placePlan =
cabinet_1_out_planning
        elif (self.pickoutIdx == 2)   :   self.placePlan =
cabinet_2_out_planning
        elif (self.pickoutIdx == 3)   :   self.placePlan =
cabinet_3_out_planning
        elif (self.pickoutIdx == 4)   :   self.placePlan =
cabinet_4_out_planning
        elif (self.pickoutIdx == 5)   :   self.placePlan =
cabinet_5_out_planning

        print("ROBOT IS PLACING EMPTY PLATE TO ORIGINAL SPACE ...!")

        jointPlanning(self.placePlan)



if __name__ == "__main__":

    robot_process = robotProcess()

    robot_process.main()
```