



คู่มือ

การควบคุมหุ่นยนต์ Dobot M1 ด้วยการเขียน
โปรแกรมภาษา Python และ Machine Vision

อนาวิน เพ็ชรบุรณิล และภวนาถ เพือกทอง

anawin.pech@gmail.com

ข้อมูลเบื้องต้นเกี่ยวกับ Dobot M1

Dobot M1 ชื่อเต็มคือ Dobot Master 1st generation robotics arm เป็นหุ่นยนต์แบบ SCARA (Selective Compliance Assembly Robot Arm) ที่มีจุดหมุน 2 จุดหมุน และมีแกนสำหรับเคลื่อนที่ในแนวตั้ง (Z-axis) เหมาะสำหรับการใช้งานในอุตสาหกรรมขนาดเล็กด้วยมีความสามารถที่หลากหลาย อาทิ การหยิบจับชิ้นงาน, การพิมพ์สามมิติ, การตัดด้วยเลเซอร์ การบัดกรีแผงวงจร การผลิตบนสายพานอัตโนมัติ และความสามารถอื่นๆ ในการควบคุมหุ่นยนต์ Dobot M1 ยังรองรับการเขียนโปรแกรมควบคุมทั้งในรูปแบบ Script และการควบคุมโดยใช้กราฟิกซอฟต์แวร์ Blockly

นอกจากนี้ยังสามารถควบคุมหุ่นยนต์ Dobot M1 โดยการเขียนโปรแกรมภาษา Python เชื่อมต่อผ่าน API และ USB port สำหรับการพัฒนาโปรแกรมในการควบคุมหุ่นยนต์

สามารถสรุปจุดเด่นของหุ่นยนต์ Dobot M1 ได้ดังนี้

- ใช้งานได้อย่างง่ายดายในการควบคุมหุ่นยนต์ผ่านซอฟต์แวร์ของหุ่นยนต์โดยตรง
- มีความแม่นยำสูงในการควบคุมการเคลื่อนที่ และความเร็ว
- สามารถรองรับโหลดได้ 1.5 kg และมีอัตราในการทำงานซ้ำ ประมาณ 0.02 mm
- รองรับการเชื่อมต่อผ่าน I/O (Input/Output) เพื่อการพัฒนาต่อยอดในขั้นต่อไป

อย่างไรก็ตามด้วยข้อจำกัดของหุ่นยนต์แบบ SCARA ทำให้ DobotM1 มีความสามารถในการรับน้ำหนักได้ต่ำ และมีระยะในการเคลื่อนที่ที่จำกัดตามมุมของการหมุนแขนของหุ่นยนต์ จึงมีความจำเป็นอย่างยิ่งที่ต้องทราบส่วนประกอบของหุ่นยนต์ (Hardware) และพื้นที่ทำงาน (Workspace)

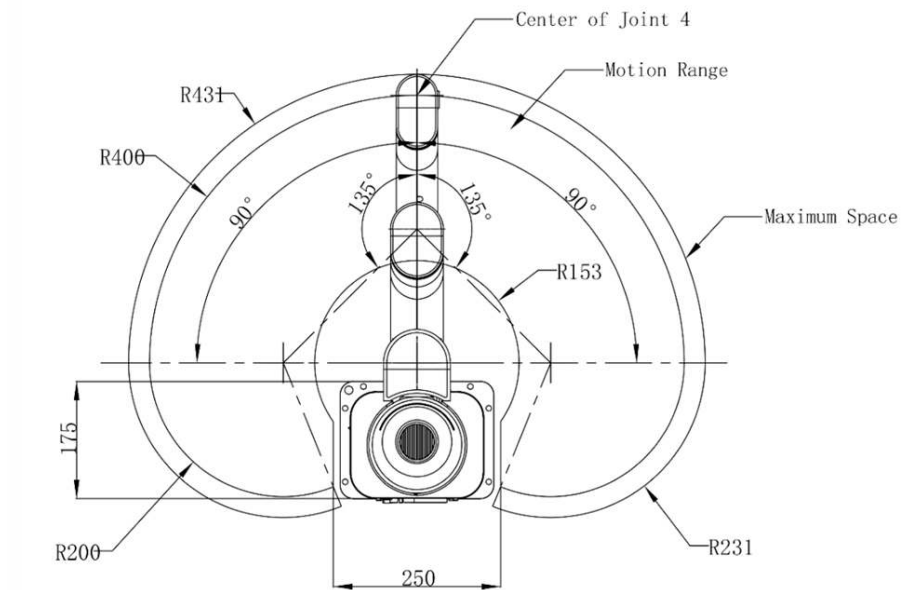
หุ่นยนต์ Dobot M1 มีส่วนประกอบ ดังนี้

- ฐานของหุ่นยนต์ (Base) ขนาด 250 x 175 mm
- แกนแนวตั้ง (Z-axis)
- แขนด้านใน (Rear arm)
- แขนด้านนอก (Forearm)

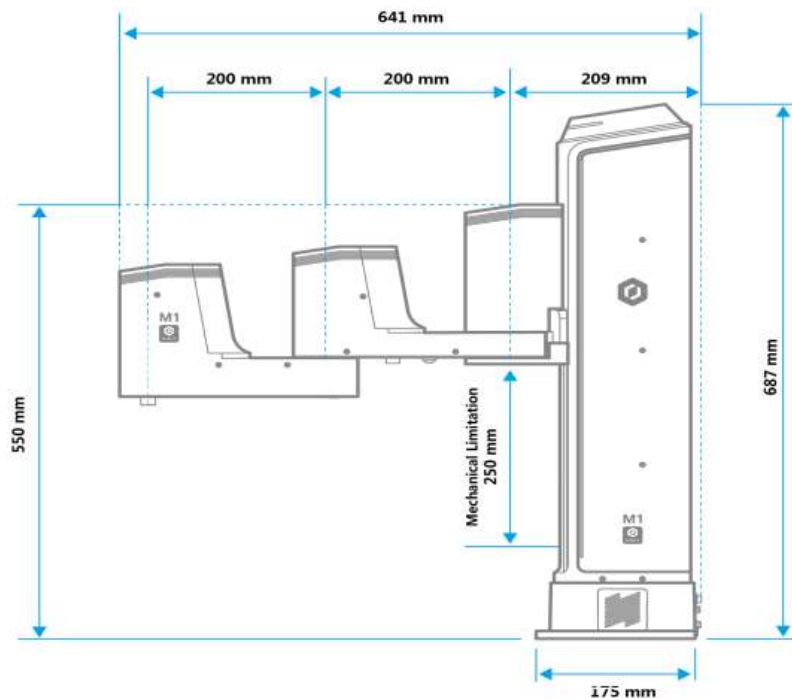


รูปที่ 1 ส่วนประกอบของหุ่นยนต์ Dobot M1

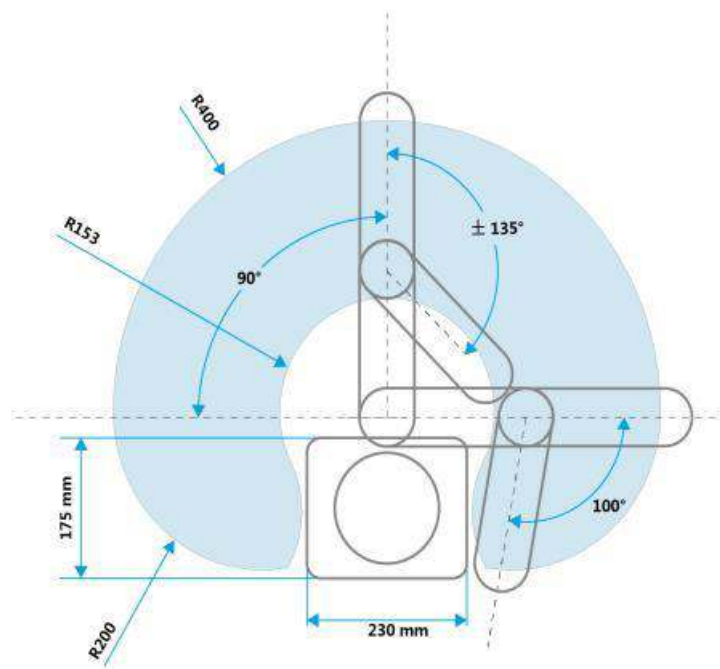
ส่วนต่อไปคือพื้นที่ทำงาน (Working space) ของหุ่นยนต์ Dobot M1 แสดงได้ดังรูปที่ 2



รูปที่ 2 Working space หุ่นยนต์ Dobot M1



รูปที่ 3 ขนาดของหุ่นยนต์ Dobot M1



รูปที่ 4 Workspace และข้อจำกัดในพื้นที่การทำงาน

พื้นฐานการควบคุม Dobot M1 ผ่าน Python-API

อย่างที่ได้อธิบายไว้ว่าการควบคุมหุ่นยนต์ Dobot M1 สามารถทำได้หลากหลายวิธี หากต้องการควบคุมหุ่นยนต์ผ่านการเขียนคำสั่งด้วยโปรแกรมภาษา Python จะต้องทำการควบคุมผ่าน API (Application Programming Interface) โดยผู้ใช้งานสามารถเขียนโปรแกรมผ่านทาง IDE เช่น Visual Studio Code เพื่อส่งการให้หุ่นยนต์ทำงานผ่าน API

IDE หรือ Integrated Development Environment เป็นเครื่องมือที่นักพัฒนาโปรแกรมใช้ในการเขียนโปรแกรม โดยอาจมีส่วนประกอบหลายส่วน เช่น ส่วนที่ใช้ในการเขียน Code (text editor) ส่วนที่ใช้ในการตรวจสอบและแก้ไขโปรแกรม (Debugging) ส่วนที่ใช้แสดงผล และส่วนขยายต่างๆ สำหรับการเขียนโปรแกรมภาษา Python จะแนะนำให้ใช้การเขียนไฟล์นามสกุล .py บน Visual Studio Code หรือ VS Code

ต่อไปนี้จะแนะนำขั้นตอนในการจัดเตรียม Environment และติดตั้งแพ็คเกจที่จำเป็นในการเขียนโปรแกรมภาษา Python เพื่อควบคุมหุ่นยนต์ Dobot M1 ดังนี้

1. ติดตั้ง VS Code สามารถติดตั้งได้ที่ <https://code.visualstudio.com/download> โดยไม่มีค่าใช้จ่าย
2. ติดตั้งส่วนขยาย (EXTENSIONS) บน VS Code ได้แก่ Python และ Pylance
3. สร้างโฟลเดอร์แยกสำหรับโปรเจกต์ และเตรียม Virtual Environment เฉพาะสำหรับโปรเจกต์ ศึกษาวิธีการเตรียม Environment ได้ที่ <https://code.visualstudio.com/docs/python/environments>
4. ติดตั้งแพ็คเกจที่จำเป็นบน Virtual Environment ผ่าน Command prompt (cmd) ดังนี้
 - pip install opencv-python
 - pip install matplotlib
 - pip install scipy

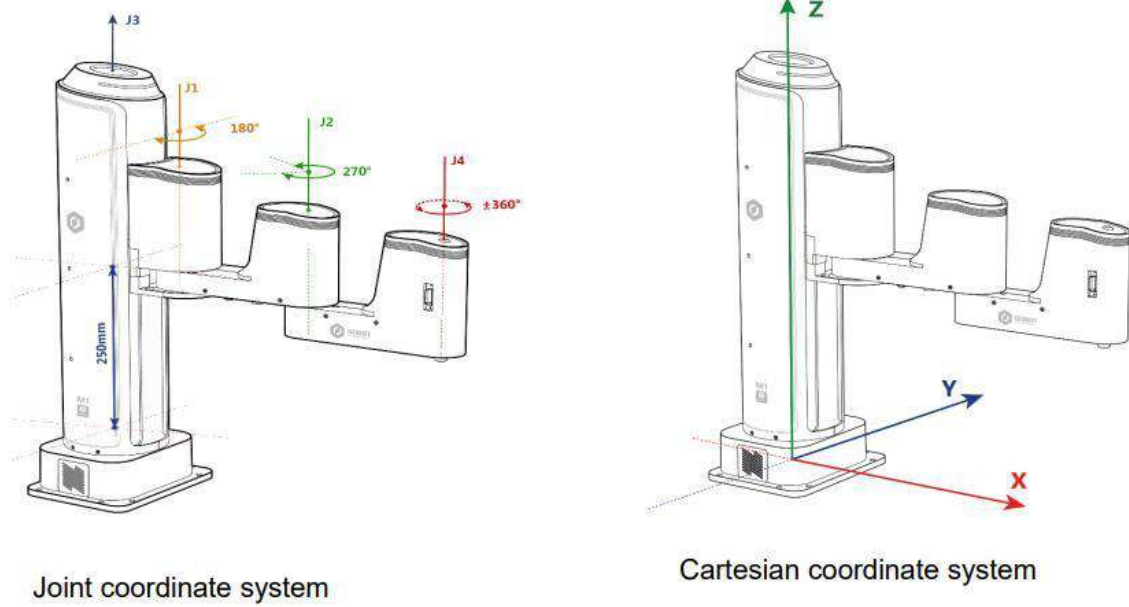
พื้นฐานการควบคุม Dobot M1 ผ่าน M1Studio

M1 Studio เป็นซอฟต์แวร์ที่ใช้ในการควบคุมหุ่นยนต์ Dobot M1 โดยสามารถควบคุมหุ่นยนต์ผ่านวิธีการต่าง ๆ ดังนี้

- **ควบคุมผ่าน Operation panel** ของซอฟต์แวร์ โดยสามารถควบคุมแบบ Joint (J1, J2, J3, J4) และควบคุมแบบพิกัด Cartesian (X, Y, Z, R) โดยสามารถตรวจสอบค่าพิกัดปัจจุบันได้ที่ Operation panel

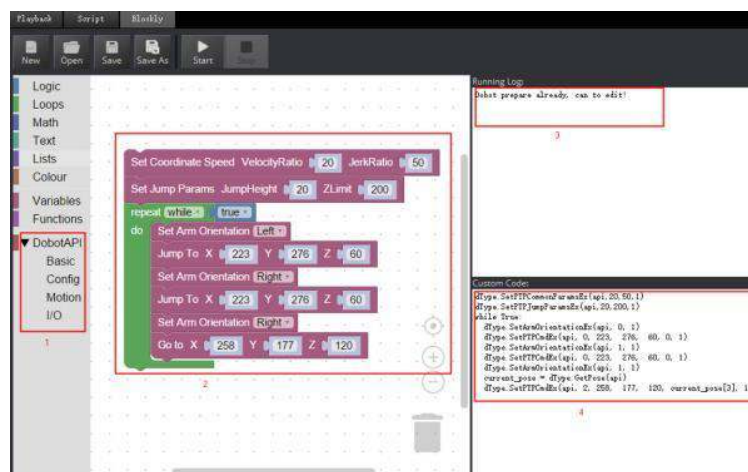


รูปที่ 5 การควบคุมผ่าน Operation panel



รูปที่ 6 ระบบพิกัดแบบ Joint และ Cartesian

- ควบคุมผ่าน Blockly เป็นแพลตฟอร์มการเขียนโปรแกรมแบบกราฟิกที่มีพื้นฐานมาจาก Google Blockly มีลักษณะเป็น Puzzle format ที่สามารถเข้าใจได้ง่าย



รูปที่ 7 การใช้ Blockly

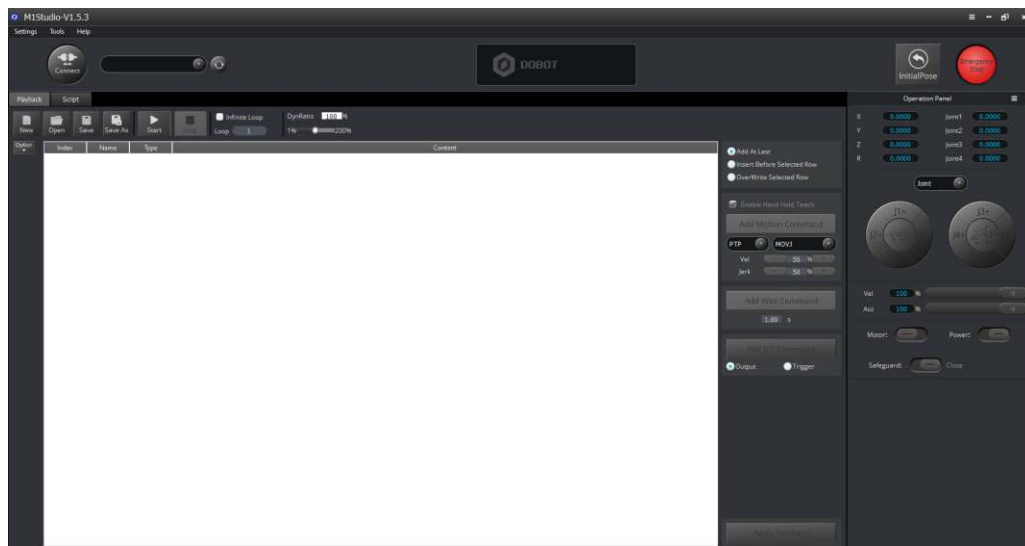
5. การให้หุ่นยนต์กลับไปตำแหน่งเริ่มต้น (Homing point) สามารถทำได้โดยการควบคุมผ่าน Operation Panel โดยกำหนดระบบพิกัด Cartesian ดังนี้ (400.0, 0.0, 234.0, 0.0) หรือไปที่แถบ Tools > Home



รูปที่ 9 ตำแหน่ง Homing point

6. หากต้องการหยุดการเชื่อมต่อกับหุ่นยนต์ให้กดปุ่ม Disconnect

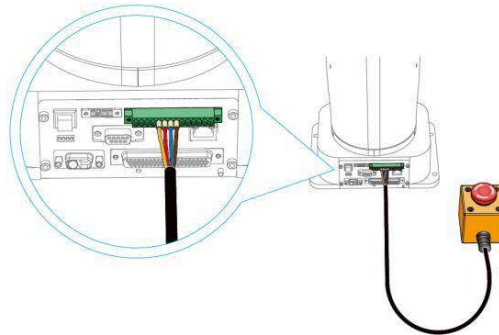
หมายเหตุ ในการควบคุมหุ่นยนต์ผ่าน Python API บน VS Code ให้ปิดการเชื่อมต่อระหว่างหุ่นยนต์กับ M1 Studio ทุกครั้ง



รูปที่ 10 หน้าต่างโปรแกรม M1 Studio ที่ไม่มีการเชื่อมต่อผ่าน USB และไม่มีการเชื่อมต่อกับหุ่นยนต์

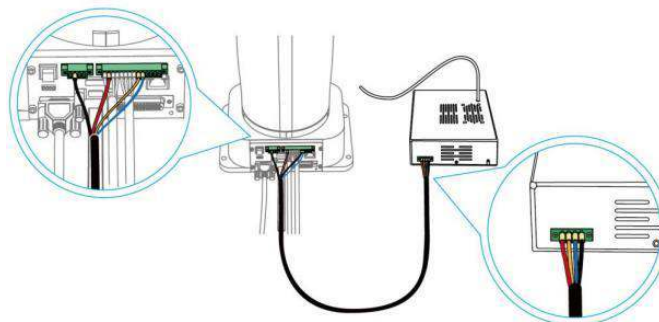
สิ่งที่ต้องเตรียมก่อนการควบคุมหุ่นยนต์ผ่าน Python API

1. ตรวจสอบว่า Switch ของหุ่นยนต์เปิด (Turn on switch)
2. ตรวจสอบว่าหุ่นยนต์ติดตั้ง Emergency Stop เรียบร้อย



รูปที่ 11 การติดตั้ง Emergency Stop

3. ตรวจสอบว่าหุ่นยนต์มีการติดตั้งปั๊มลมที่ถูกต้องตามคู่มือ Power I/O หมายเลข 17 และ Control I/O หมายเลข 18 และมีการติดตั้งมือจับชิ้นงานหรือหัวดูดชิ้นงาน



รูปที่ 12 การติดตั้ง Air Pump

4. เชื่อมต่อระหว่างคอมพิวเตอร์กับหุ่นยนต์ผ่าน USB และตรวจสอบ Port การเชื่อมต่อโดยใช้ M1 Studio
5. ติดตั้ง VS Code และจัดเตรียม Virtual Environment ที่จำเป็น
6. Download Folder ที่มีชื่อว่า **DobotM1_Python** โดยสามารถ download ได้ที่ <https://drive.google.com/drive/folders/1lhKSaLg9wBDDt4NMKbpBNiKkhVH7-HMv?usp=sharing>

7. หากต้องการใช้งานหุ่นยนต์สำหรับงานด้าน Machine Vision ให้เชื่อมต่อกล้องเข้ากับคอมพิวเตอร์ที่กำลังเชื่อมต่อกับหุ่นยนต์

สำหรับไฟล์ที่อยู่ในโฟลเดอร์ DobotM1_Python จะประกอบด้วยไฟล์สำคัญที่ใช้ในการควบคุมหุ่นยนต์ และไฟล์ที่ช่วยในการควบคุมหุ่นยนต์ผ่าน API สำหรับไฟล์สำคัญที่จำเป็นต้องรู้จักมีดังนี้

- **DobotAPI.py**

เป็น Code ที่รวบรวมฟังก์ชันการเชื่อมต่อผ่าน API

- **DobotControl.py**

เป็น Code ที่เขียนโปรแกรมสำหรับเรียกใช้งาน Method ภายใน Module เป็นเทคนิคการเขียนโปรแกรมเชิงวัตถุ (Object Oriented Programing: OOP) เพื่อความสะดวกในการเรียกใช้งานคำสั่งควบคุม

- **DobotTypes.py**

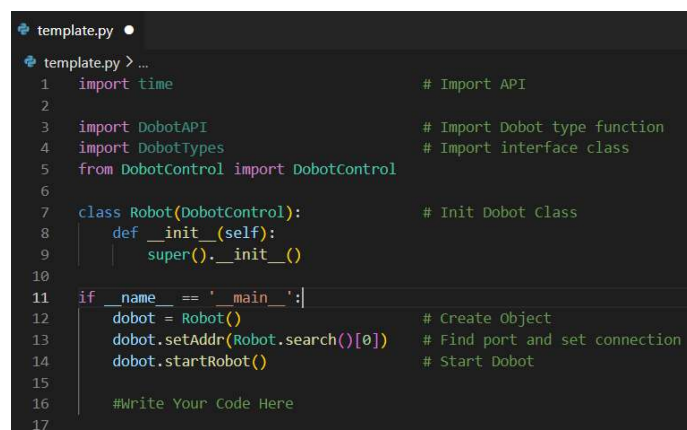
เป็น Code ที่เขียนโปรแกรมสำหรับเรียกใช้งาน Method ภายใน Module สำหรับการควบคุมหุ่นยนต์ของ DobotControl.py

- **template.py**

เป็น Code เริ่มต้นในการเขียนโปรแกรมเพื่อควบคุมหุ่นยนต์ โดยมีการเรียกใช้งาน Module ภายใน DobotControl.py

- **color_regconition.py**

เป็น Code ที่เขียนโปรแกรมสำหรับเรียกใช้งาน Method ภายใน CModule สำหรับการทำการตรวจจับวัตถุด้วยสี (Color Detection) โดยรับค่าจากกล้อง Webcam



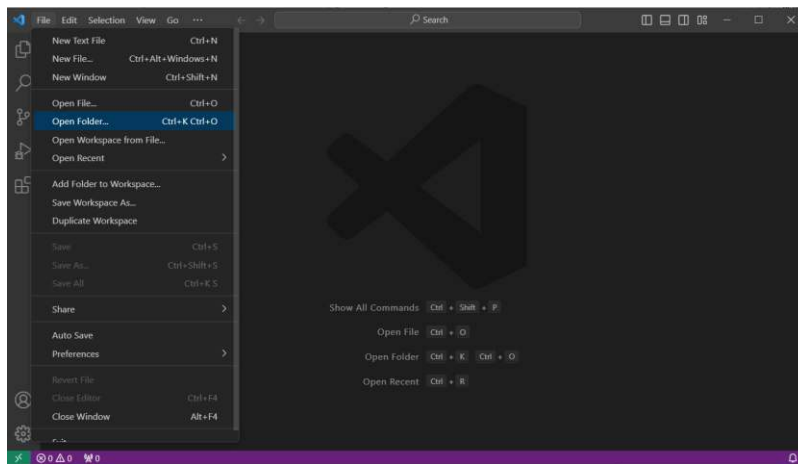
```
template.py
1  import time                    # Import API
2
3  import DobotAPI                # Import Dobot type function
4  import DobotTypes              # Import interface class
5  from DobotControl import DobotControl
6
7  class Robot(DobotControl):      # Init Dobot Class
8      def __init__(self):
9          super().__init__()
10
11 if __name__ == '__main__':
12     dobot = Robot()              # Create Object
13     dobot.setAddr(Robot.search()[0]) # Find port and set connection
14     dobot.startRobot()           # Start Dobot
15
16     #Write Your Code Here
17
```

รูปที่ 13 ไฟล์ template.py

เริ่มต้นการควบคุมหุ่นยนต์ผ่าน Python API

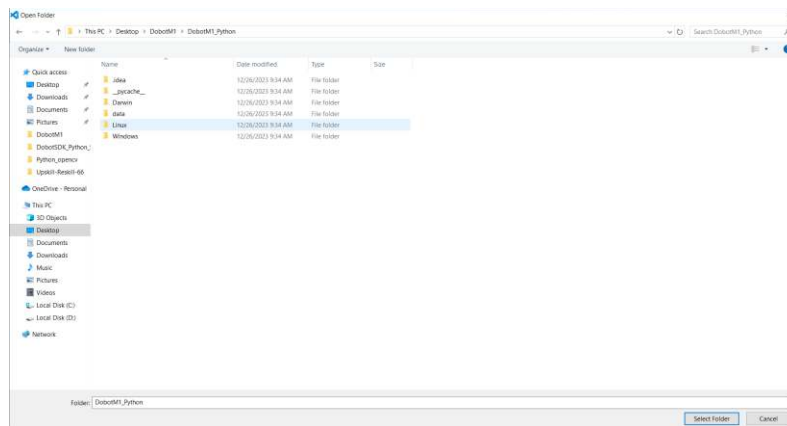
ในหัวข้อนี้จะเป็นการเริ่มต้นการเชื่อมต่อหุ่นยนต์ผ่าน Python API และการเรียกใช้งานฟังก์ชันพื้นฐานสำหรับควบคุมหุ่นยนต์ รวมถึงการอธิบายหลักการทำงานของฟังก์ชัน โดยมีขั้นตอนดังต่อไปนี้

1. เปิดใช้งาน VS Code เปิดโฟลเดอร์ DobotM1_Python โดยสามารถทำได้โดยไปที่แถบ File > Open Folder...



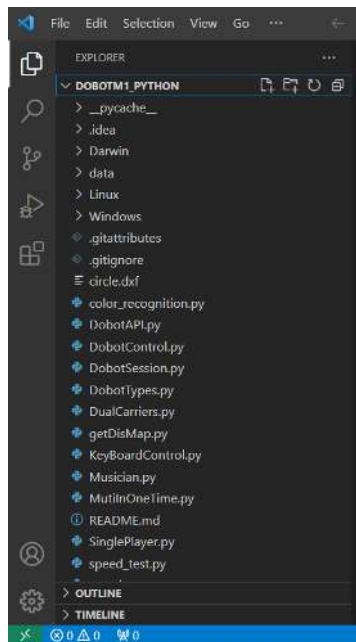
รูปที่ 14 วิธีการเปิดใช้งานโฟลเดอร์

2. หาดำแหน่งที่เก็บโฟลเดอร์ DobotM1_Python จากนั้นคลิกเข้าไปในโฟลเดอร์ > กด Select Folder



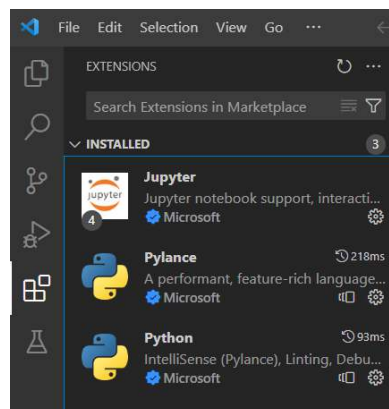
รูปที่ 15 เลือกโฟลเดอร์ DobotM1_Python

3. เมื่อเลือกโฟลเดอร์แล้วจะปรากฏหน้าต่าง DobotM1_Python ตรงบริเวณแถบ EXPLORER



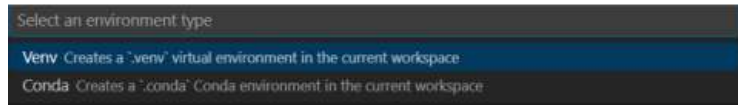
รูปที่ 16 โฟลเดอร์ DobotM1_Python ใน VS Code

4. ติดตั้ง EXTENSIONS ได้แก่ Python และ Pylance ภายใน VS Code

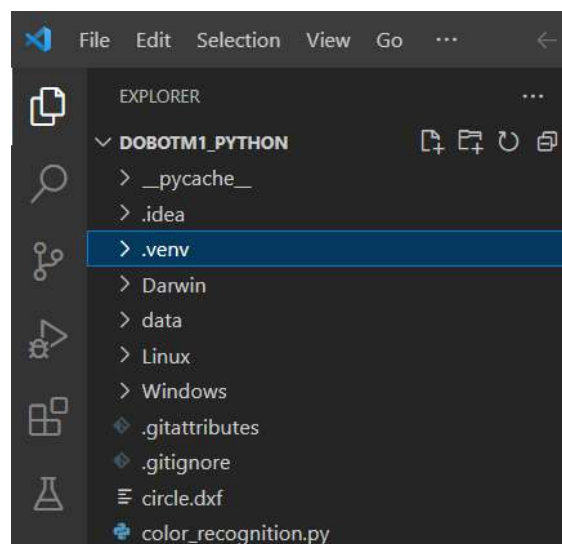


รูปที่ 17 EXTENSIONS ที่จำเป็น ได้แก่ Python และ Pylance


5. สร้าง Virtual Environment ให้ทำการการเปิด Command Palette โดยกดปุ่ม Ctrl + Shift + P พิมพ์ค้นหา **Python: Create Environment** จากนั้นเลือก Venv ซอฟต์แวร์จะทำการสร้างและ Activate Environment ให้โดยอัตโนมัติ

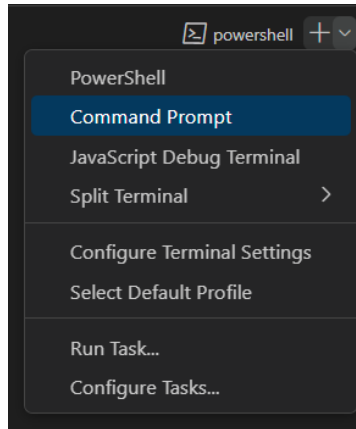


รูปที่ 18 สร้าง Virtual Environment



รูปที่ 19 โฟลเดอร์ Virtual Environment (.venv) ที่สร้างสำเร็จ

6. กดที่ปุ่ม Toggle panel  (Ctrl + J) เพื่อแสดงแถบ TERMINAL ให้เปลี่ยนจาก Power shell เป็น Command Prompt (รูปที่ 20) จะขึ้นหน้าต่าง Command Prompt ให้กดปิด (Kill) หน้าต่าง Power Shell (รูปที่ 21)

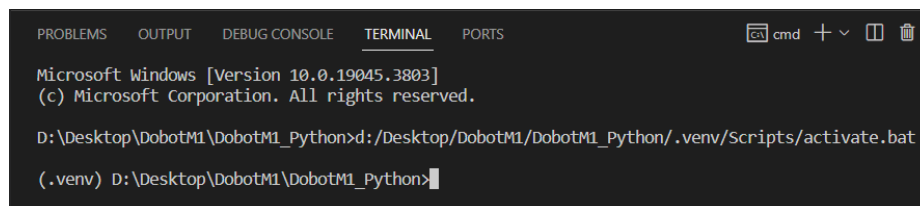


รูปที่ 20 เปิด Command Prompt



รูปที่ 21 ปิดหน้าต่าง Power Shell

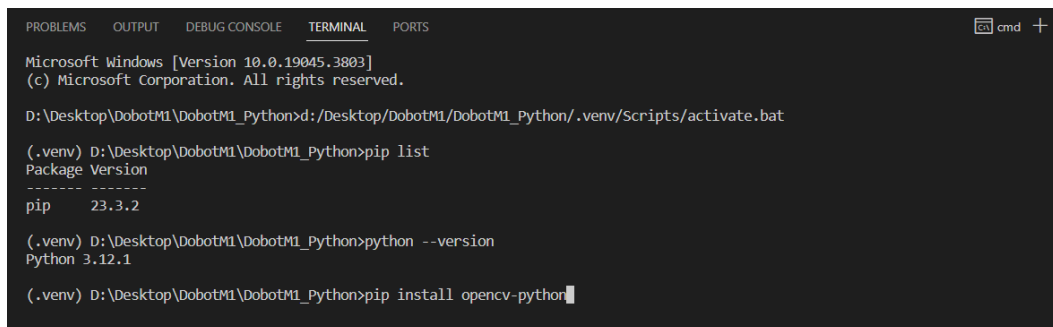
7. หากการสร้าง Virtual Environment เสร็จสมบูรณ์จะปรากฏ (.venv) บน Command Line



รูปที่ 22 Virtual Environment บน Command Line

8. ติดตั้งแพ็คเกจบน Virtual Environment ด้วย Command Prompt พิมพ์คำสั่งต่อไปนี้ เมื่อพิมพ์เสร็จแต่ละคำสั่งให้กด Enter

- python --version (เป็นการตรวจสอบ Python version)
- pip list (แสดงรายชื่อแพ็คเกจที่มีการติดตั้งไว้แล้ว)
- pip install opencv-python (ติดตั้งแพ็คเกจ OpenCV และ Numpy)
- pip install scipy (ติดตั้งแพ็คเกจ Scipy)
- pip install matplotlib (ติดตั้งแพ็คเกจ Matplotlib)



```
Microsoft Windows [Version 10.0.19045.3803]
(c) Microsoft Corporation. All rights reserved.

D:\Desktop\DobotM1\DobotM1_Python>D:\Desktop\DobotM1\DobotM1_Python/.venv/Scripts/activate.bat

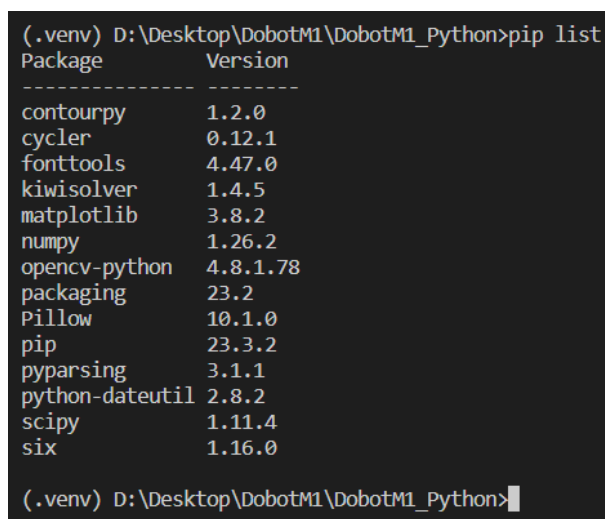
(.venv) D:\Desktop\DobotM1\DobotM1_Python>pip list
Package Version
-----
pip      23.3.2

(.venv) D:\Desktop\DobotM1\DobotM1_Python>python --version
Python 3.12.1

(.venv) D:\Desktop\DobotM1\DobotM1_Python>pip install opencv-python
```

รูปที่ 23 การตรวจสอบและติดตั้งแพ็คเกจบน Command Line

รูปที่ 23 คำสั่ง `pip list` แสดงการติดตั้งแพ็คเกจ `pip` ใน version 23.3.2 โดยมีไว้สำหรับเรียกการติดตั้งแพ็คเกจอื่นต่อไป คำสั่ง `python --version` แสดง version ของ python ในที่นี้คือ Python version 3.12.1 คำสั่ง `pip install opencv-python` เป็นการเรียกการติดตั้งแพ็คเกจ OpenCV และเมื่อติดตั้งแพ็คเกจทั้งหมดที่ต้องการแล้วให้ตรวจสอบการติดตั้งใหม่อีกครั้งด้วยคำสั่ง `pip list`



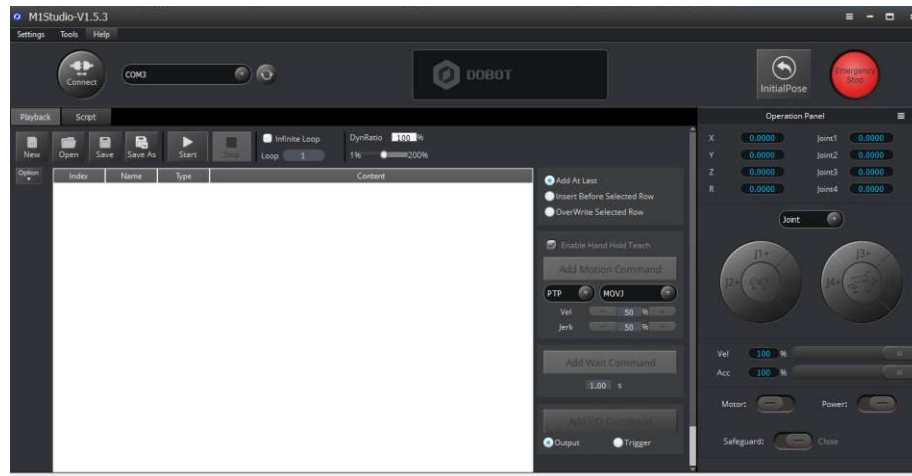
```
(.venv) D:\Desktop\DobotM1\DobotM1_Python>pip list
Package Version
-----
contourpy 1.2.0
cycler     0.12.1
fonttools  4.47.0
kiwisolver 1.4.5
matplotlib 3.8.2
numpy      1.26.2
opencv-python 4.8.1.78
packaging  23.2
Pillow     10.1.0
pip        23.3.2
pyparsing  3.1.1
python-dateutil 2.8.2
scipy      1.11.4
six        1.16.0

(.venv) D:\Desktop\DobotM1\DobotM1_Python>
```


รูปที่ 24 ตรวจสอบการติดตั้งแพ็คเกจที่จำเป็น

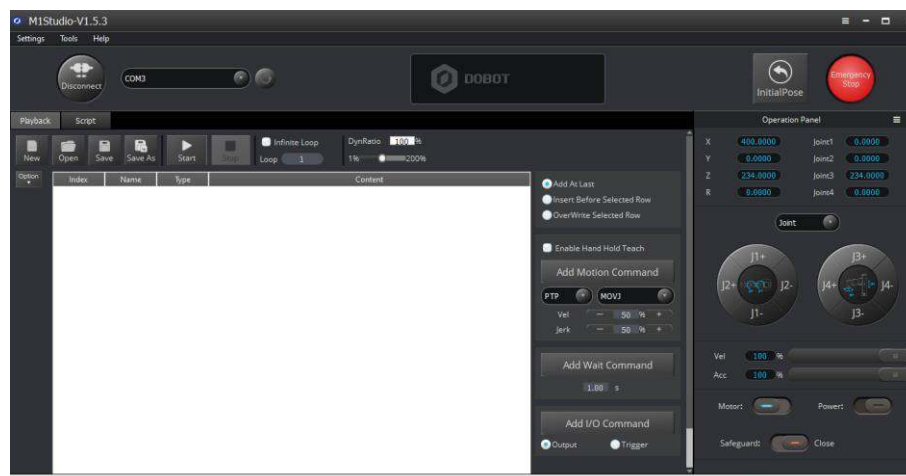
หากดำเนินการจนครบข้อ 8. แล้วจะถือว่าการจัดเตรียม Environment ที่จำเป็นสำหรับการควบคุมหุ่นยนต์ Dobot M1 เสร็จสมบูรณ์ขั้นตอนถัดไปจะทำการเชื่อมต่อหุ่นยนต์เข้ากับคอมพิวเตอร์ที่ได้มีการเตรียมการไว้ก่อนหน้า

9. เปิดหุ่นยนต์โดยการกดปุ่ม ON/OFF Switch ค้างไว้ 3 วินาที เสียบสาย USB จากหุ่นยนต์เข้าสู่ USB Port ของคอมพิวเตอร์ ไฟ LED สีเขียว (ดวงที่ 2 จากซ้ายมือ) จะกระพริบ จากนั้นให้เปิดโปรแกรม M1Studio จะปรากฏหมายเลข Port ดังรูป



รูปที่ 25 เปิดใช้งานโปรแกรม M1Studio

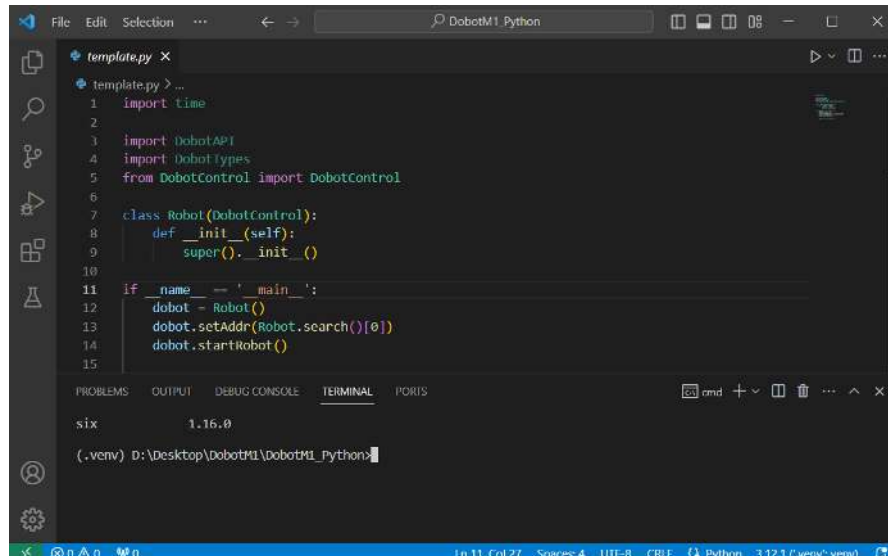
10. กดปุ่ม Connect  เพื่อเชื่อมต่อเข้ากับหุ่นยนต์ ตรวจสอบว่าหุ่นยนต์อยู่ที่ตำแหน่ง Homing point หรือไม่ โดยดูที่ Operation panel จะมีค่าพิกัด Cartesian (X, Y, Z, R) เท่ากับ (400.0, 0.0, 234.0, 0.0) หรือมีพิกัดของ Joint เท่ากับ (0.0, 0.0, 234.0, 0.0) ดังรูปที่ 26



รูปที่ 26 โปรแกรม M1Studio แสดงตำแหน่งของหุ่นยนต์ที่ Homing point

11. ขั้นตอนต่อมาคือการเชื่อมต่อหุ่นยนต์ผ่าน Python API ให้ทำการปิดการเชื่อมต่อกับ M1Studio โดย

การกดปุ่ม Disconnect  จากนั้นไปที่ VS Code เปิดไฟล์ **template.py** กด Run 



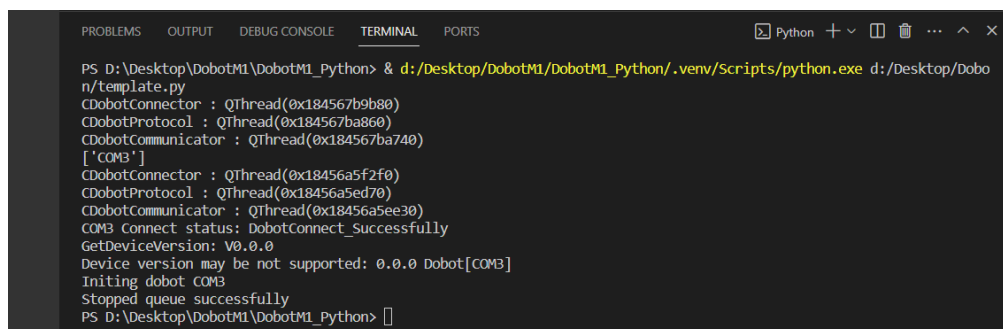
```

1 import time
2
3 import DobotAPI
4 import DobotTypes
5 from DobotControl import DobotControl
6
7 class Robot(DobotControl):
8     def __init__(self):
9         super().__init__()
10
11 if __name__ == '__main__':
12     robot = Robot()
13     robot.setAddr(Robot.search()[0])
14     robot.startRobot()
15

```

รูปที่ 27 เปิดไฟล์ template.py

หากเชื่อมต่อสำเร็จจะปรากฏข้อความบน Python Shell (เปลี่ยนจาก Command Prompt กลับเป็น Python Shell) ดังนี้



```

PS D:\Desktop\DobotM1\DobotM1_Python> & d:/Desktop/DobotM1/DobotM1_Python/.venv/Scripts/python.exe d:/Desktop/DobotM1/DobotM1_Python/template.py
CDobotConnector : QThread(0x184567b9b80)
CDobotProtocol : QThread(0x184567ba860)
CDobotCommunicator : QThread(0x184567ba740)
['COM3']
CDobotConnector : QThread(0x18456a5f2f0)
CDobotProtocol : QThread(0x18456a5ed70)
CDobotCommunicator : QThread(0x18456a5ee30)
COM3 Connect status: DobotConnect_Successfully
GetDeviceVersion: V0.0.0
Device version may be not supported: 0.0.0 Dobot[COM3]
Initing dobot COM3
Stopped queue successfully
PS D:\Desktop\DobotM1\DobotM1_Python>

```

รูปที่ 28 เชื่อมต่อ Python API กับหุ่นยนต์สำเร็จ

12. ในหัวข้อนี้จะเป็นการเริ่มควบคุมหุ่นยนต์โดยการเรียกใช้ฟังก์ชัน ที่มีการเขียนเป็น Object ไว้บน Class ที่อยู่ในไฟล์ **DobotControl.py** โดยจะทำการเขียนเพื่อเรียกใช้งานในไฟล์ **template.py** ในเบื้องต้นจะให้เริ่มต้นเขียนได้ที่บรรทัดที่ 17 ได้ข้อความ **#Write Your Code Here** สำหรับ ฟังก์ชันที่มีการเตรียมไว้ให้สำหรับการควบคุมหุ่นยนต์แสดง ดังตารางที่ 1

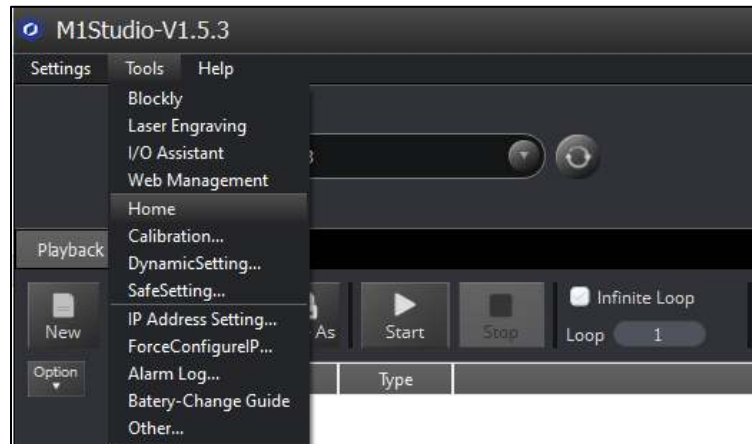
ตารางที่ 1 ฟังก์ชันพื้นฐานในการควบคุมหุ่นยนต์ Dobot M1

ฟังก์ชัน	ส่วนประกอบ	คำอธิบาย
moveTo(x,y,z,r)	x, y, z, r คือพิกัดในระบบ cartesian	เป็นคำสั่งสำหรับเคลื่อนที่หุ่นยนต์ ไปที่พิกัดที่ต้องการ (Absolute Position)
moveInc(dx,dy,dz,dr)	dx, dy, dz, dr คือการกำหนด ระยะการเคลื่อนที่จากจุดเดิม ในหน่วย mm ตามแกนนั้นๆ	เป็นคำสั่งสำหรับเคลื่อนที่หุ่นยนต์ ไปจากจุดเดิมตามระยะที่กำหนด (Relative Position)
setPump(Power I/O, Control I/O)	หมายเลข Power I/O = 17 หมายเลข Control I/O = 18	เป็นคำสั่งการเปิดทำงานของ Pump
resetPump(Power I/O, Control I/O)	หมายเลข Power I/O = 17 หมายเลข Control I/O = 18	เป็นคำสั่งการปิดการทำงานของ Pump
suck()		เป็นคำสั่งสำหรับสั่งให้ปั๊มลมทำการดูด เพื่อหยิบชิ้นงาน
unsuck()		เป็นคำสั่งสำหรับสั่งให้ปั๊มหยุดการดูด เพื่อปล่อยชิ้นงาน
clearAlarm()		เป็นคำสั่งในการปิดการแจ้งเตือนของหุ่นยนต์
Time.sleep(Secs)	Secs คือ จำนวนวินาที	เป็นคำสั่งสำหรับหน่วงการทำงานของหุ่นยนต์ในหน่วย วินาที

ตัวอย่างการใช้งานฟังก์ชัน เช่น

- สั่งให้หุ่นยนต์เคลื่อนไปที่ตำแหน่ง Homing point ด้วย moveTo()
moveTo(400.0, 0.0, 234.0, 0.0)
- สั่งให้หุ่นยนต์ขยับลง 10 cm (100 mm) ตามแนวแกน Z ด้วย moveInc()
moveInc(0.0, 0.0, -100.0, 0.0)

13. การปิดใช้งานหุ่นยนต์ให้เริ่มจาก ใช้คำสั่งผ่าน Python API ให้หุ่นยนต์เคลื่อนที่ไปที่ตำแหน่ง Homing Point หรือใช้โปรแกรม M1Studio โดยปิด VS Code กด Connect เพื่อเชื่อมต่อจากนั้นไปที่แถบ Tools > Home รอจนกระทั่งหุ่นยนต์เคลื่อนไปที่ Homing point จากนั้นกด Disconnect



รูปที่ 29 การตั้ง Homing Point ใน M1Studio

14. กดปุ่ม ON/OFF Switch ค้างไว้ 3 วินาทีแล้วจึงปล่อย รอจนกระทั่งไฟ LED ดับลงจากนั้นจึงถอดสาย USB ออกจากคอมพิวเตอร์ ถอดปลั๊กไฟ

หัวข้อนี้ได้อธิบายขั้นตอนการเตรียมการเพื่อควบคุมหุ่นยนต์ผ่าน API โดยเริ่มตั้งแต่การติดตั้งซอฟต์แวร์ที่จำเป็น การเตรียม Environment การติดตั้งแพ็คเกจ และฟังก์ชันพื้นฐานในการควบคุมหุ่นยนต์ สำหรับหัวข้อถัดไปจะเป็นตัวอย่างการใช้งานฟังก์ชันเพื่อควบคุมหุ่นยนต์ในกรณีต่างๆ เช่น การเคลื่อนที่จากการรับคำสั่งด้วยการป้อนค่า (Input) การหยุดและปล่อยชิ้นงานตามตำแหน่งที่กำหนด และตัวอย่างอื่นๆ เพื่อให้เข้าใจวิธีการควบคุมหุ่นยนต์มากยิ่งขึ้น

ตัวอย่างการควบคุมหุ่นยนต์

ตัวอย่างที่ 1 การเคลื่อนที่ของหุ่นยนต์พื้นฐานด้วยโปรแกรมภาษา Python โดยกำหนดพิกัดในการเคลื่อนที่ให้เขียนคำสั่งควบคุมหุ่นยนต์ให้เคลื่อนที่ไปตามพิกัด ดังนี้

ตำแหน่ง	พิกัด (X, Y, Z, R)
A	(240.0, 120.0, 60.0, 0.0)
B	(160.0, 60.0, 60.0, 0.0)
C	(160.0, 60.0, 41.0, 30.0)
Homing point	(400.0, 0.0, 234.0, 0.0)

ส่วนที่ 1 คำสั่งสำหรับเชื่อมต่อ และเรียกใช้งานฟังก์ชันควบคุม

```
[1] import time                                # เรียกใช้ API
[2] import DobotAPI                            # เรียกใช้ฟังก์ชัน Dobot API
[3] import DobotTypes                          # เรียกใช้ฟังก์ชัน DobotTypes
[4] from DobotControl import DobotControl      # เรียกใช้ Module : Dobot Control
[5] class Robot(DobotControl):                 # กำหนดคลาสเริ่มต้นเป็น Dobot Control
[6]     def __init__(self):                     # กำหนด Method ในการสร้างวัตถุ
[7]         super().__init__()
[8] if __name__ == '__main__':                  # ตรวจสอบว่าเป็น Module หลัก
[9]     dobot = Robot()                          # สร้างวัตถุ (Object) คือ Dobot
[10]    dobot.setAddr(Robot.search())[0])        # หา port และทำการเชื่อมต่อ
[11]    dobot.startRobot()                       # เริ่มการใช้งาน Dobot
```

ส่วนที่ 2 เริ่มต้นการเขียนคำสั่งควบคุมหุ่นยนต์

คำสั่ง moveTo(X, Y, Z, R) สำหรับควบคุมการเคลื่อนที่โดยกำหนดจากพิกัด

```
[12] dobot.moveTo(240.0,120.0,60.0,0.0)
[13] dobot.moveTo(160.0,60.0, 60.0,0.0)
[14] dobot.moveTo(160.0,60.0, 41.0,30.0)
[15] dobot.moveTo(400.0, 0.0, 234.0,0.0)      # Homing point
```

ตัวอย่างที่ 2 การเคลื่อนที่ของหุ่นยนต์พื้นฐานด้วยโปรแกรมภาษา Python โดยกำหนดระยะในการเคลื่อนที่ให้เขียนคำสั่งควบคุมหุ่นยนต์โดยรับค่า (Input) จากคีย์บอร์ดให้เคลื่อนที่ โดยกำหนดชุดคำสั่ง ดังนี้

ชื่อคำสั่ง	ผลลัพธ์
up	เคลื่อนที่ขึ้นจากตำแหน่งเดิม 10 เซนติเมตร
down	เคลื่อนที่ลงจากตำแหน่งเดิม 10 เซนติเมตร
left	เคลื่อนที่ไปทางซ้ายจากตำแหน่งเดิม 10 เซนติเมตร
right	เคลื่อนที่ไปทางขวาจากตำแหน่งเดิม 10 เซนติเมตร
front	เคลื่อนที่ไปข้างหน้าจากตำแหน่งเดิม 10 เซนติเมตร
back	เคลื่อนที่ถอยหลังจากตำแหน่งเดิม 10 เซนติเมตร
home	เคลื่อนที่ไปที่ตำแหน่งเริ่มต้น (Homing point)

ส่วนที่ 1 คำสั่งสำหรับเชื่อมต่อ และเรียกใช้งานฟังก์ชันควบคุม

```
[1] import time
[2] import DobotAPI
[3] import DobotTypes
[4] from DobotControl import DobotControl
[5] class Robot(DobotControl):
[6]     def __init__(self):
[7]         super().__init__()
[8] if __name__ == '__main__':
[9]     dobot = Robot()
[10]     dobot.setAddr(Robot.search()[0])
[11]     dobot.startRobot()
```

ส่วนที่ 2 เริ่มต้นการเขียนคำสั่งควบคุมหุ่นยนต์

```
[12] cmd = input("Please Enter Command: ")
[13] if cmd == "up" :
[14]     dobot.moveInc(0.0,0.0,100.0,0.0)
[15] elif cmd == "down" :
[16]     dobot.moveInc(0.0,0.0,-100.0,0.0)
[17] elif cmd == "front" :
[18]     dobot.moveInc(100.0,0.0,0.0,0.0)
[19] elif cmd == "back" :
[20]     dobot.moveInc(-100.0,0.0,0.0,0.0)
[21] elif cmd == "left" :
[22]     dobot.moveInc(0.0,100.0,0.0,0.0)
[23] elif cmd == "right" :
[24]     dobot.moveInc(0.0,-100.0,0.0,0.0)
[25] elif cmd == "home" :
[26]     dobot.moveTo(400.0, 0.0, 234.0,0.0)
[27] else :
[28]     pass                # หากป้อนค่าไม่ตรงให้ข้ามการทำงานไปด้วยคำสั่ง pass
```

ตัวอย่างที่ 3 การเขียนคำสั่งควบคุมการใช้ปั๊มลมด้วยโปรแกรมภาษา Python

ให้เขียนคำสั่งเปิดการใช้งานปั๊มลม เขียนคำสั่งควบคุมหัวดูด (Suction cup) โดยให้หุ่นยนต์เคลื่อนที่ลงมาดูด จากนั้นเคลื่อนที่กลับไปตำแหน่งเริ่มต้นแล้วจึงปล่อยวัตถุ จากนั้นทำการปิดการทำงานของปั๊ม

ส่วนที่ 1 คำสั่งสำหรับเชื่อมต่อ และเรียกใช้งานฟังก์ชันควบคุม

```
[1] import time
[2] import DobotAPI
[3] import DobotTypes
[4] from DobotControl import DobotControl
[5] class Robot(DobotControl):
[6]     def __init__(self):
[7]         super().__init__()
[8] if __name__ == '__main__':
[9]     dobot = Robot()
[10]     dobot.setAddr(Robot.search()[0])
[11]     dobot.startRobot()
```

ส่วนที่ 2 เริ่มต้นการเขียนคำสั่งควบคุมหุ่นยนต์

```
[12] dobot.setPump(17,18) # เปิดการทำงานของปั๊ม ให้ตรวจสอบการเชื่อมต่อ
                                Power I/O = 17, Control I/O = 18
[13] dobot.moveInc( dx, dy, dz, dr ) หรือ dobot.moveTo(x,y,z,r) # ให้แก้ไขบรรทัดนี้
[14] dobot.suck()           # คำสั่งให้หุ่นยนต์หยิบชิ้นงาน
[15] dobot.moveTo(400.0, 0.0, 234.0,0.0) # คำสั่งให้หุ่นยนต์เคลื่อนที่ไปที่ตำแหน่งเริ่มต้น
[16] time.sleep(1)         # หน่วงเวลา 1 วินาที
[17] dobot.unsuck()        # คำสั่งให้หุ่นยนต์ปล่อยชิ้นงาน
[18] dobot.resetPump(17,18) # ปิดการทำงานของปั๊ม
```


ตัวอย่างที่ 4 การเขียนคำสั่งวนซ้ำควบคุมการหยิบชิ้นงานด้วยโปรแกรมภาษา Python

จากตัวอย่างที่ 3 ให้เขียนคำสั่งแบบวนซ้ำ (Loop programming) ให้หุ่นยนต์เคลื่อนที่ลงมาหยิบวัตถุที่ตำแหน่งเดิม จากนั้นเคลื่อนที่ไปปล่อยวัตถุที่ตำแหน่งใหม่ แล้วกลับไปตำแหน่งเริ่มต้น โดยให้วนซ้ำจำนวน 5 ครั้ง

ส่วนที่ 1 คำสั่งสำหรับเชื่อมต่อ และเรียกใช้งานฟังก์ชันควบคุม

```
[1] import time
[2] import DobotAPI
[3] import DobotTypes
[4] from DobotControl import DobotControl
[5] class Robot(DobotControl):
[6]     def __init__(self):
[7]         super().__init__()
[8] if __name__ == '__main__':
[9]     dobot = Robot()
[10]     dobot.setAddr(Robot.search()[0])
[11]     dobot.startRobot()
```

ส่วนที่ 2 เริ่มต้นการเขียนคำสั่งควบคุมหุ่นยนต์

```
[12] for i in range (1,6):                                     # For loop ให้ทำงานซ้ำ 5 ครั้ง
[13]     dobot.setPump(17,18)
[14]     dobot.moveInc( dx, dy, dz, 0.0 ) หรือ dobot.moveTo(x,y,z,r) # ตำแหน่งที่หยิบชิ้นงาน
[15]     time.sleep(1)
[16]     dobot.suck()
[17]     dobot.moveInc( dx, dy, dz, 0.0 ) หรือ dobot.moveTo(x,y,z,r) # ตำแหน่งที่ปล่อยชิ้นงาน
[18]     dobot.unsuck()
[19]     dobot.moveTo(400.0,0.0,234.0,0.0)                       # กลับมาที่ตำแหน่งเริ่มต้น
[20] dobot.resetPump(17,18)
```

แบบฝึกหัดการควบคุมหุ่นยนต์

1. สั่งให้หุ่นยนต์เคลื่อนที่ไปยังพิกัดต่อไปนี้

$X = 240, Y = 120, Z = 60, R = 0$

$X = 160, Y = 60, Z = 60, R = 0$

$X = 160, Y = 60, Z = 0, R = 0$

ให้หน่วงเวลาไว้ 2 วินาที จากนั้นให้เคลื่อนที่ไปยังพิกัด

$X = 80, Y = 60, Z = 0, R = 0$

ให้หน่วงเวลาไว้ 1 วินาที จากนั้นให้เคลื่อนที่ไปยังพิกัด

$X = 80, Y = 60, Z = 60, R = 0$

$X = 80, Y = 120, Z = 60, R = 0$

$X = 240, Y = 120, Z = 60, R = 0$

ให้หน่วงเวลาไว้ 5 วินาที จากนั้นให้เคลื่อนที่ไปยังตำแหน่งเริ่มต้น (Homing point)

2. ให้เขียนคำสั่งแบบวนซ้ำ (Loop programming) ให้หุ่นยนต์เคลื่อนที่ลงมาหยิบวัตถุที่ตำแหน่งเดิม จากนั้นเคลื่อนที่ไปปล่อยวัตถุที่ตำแหน่งใหม่ โดยให้วนซ้ำจำนวน 3 ครั้ง

for ตัวแปร in range (ช่วงข้อมูล):

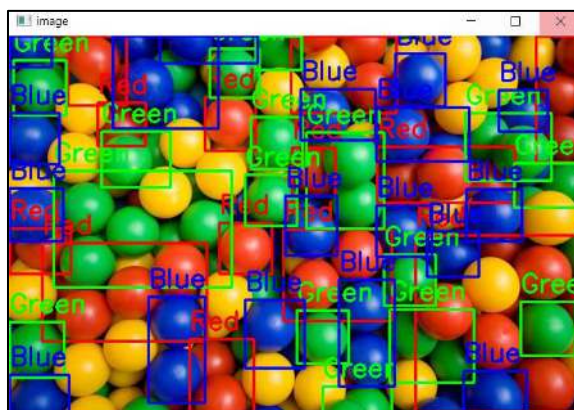
คำสั่งต่างๆ

3. ให้เขียนคำสั่งวนซ้ำ (Loop programming) ให้หุ่นยนต์เคลื่อนที่ลงมาหยิบวัตถุที่ตำแหน่งเดิม จากนั้นให้เรียงวัตถุเป็นเส้นตรง หรือเป็นแถวยาว

การใช้งาน Machine Vision กับหุ่นยนต์ Dobot M1

การมองเห็นของเครื่องจักร (Machine Vision) เป็นเทคนิคหนึ่งในการเพิ่มความสามารถของเครื่องจักรให้สามารถมองเห็นได้ โดยอาจทำงานร่วมกับระบบอัตโนมัติ (Automation System) เพื่อให้การทำงานของเครื่องจักรหรือหุ่นยนต์มีความแม่นยำมากยิ่งขึ้น เพิ่มความสามารถในการตรวจสอบความผิดพลาดที่อาจเกิดขึ้น ใช้ในการตรวจสอบ การแยกแยะ การวัดขนาด การหาตำแหน่งของวัตถุหรือชิ้นงาน และการอ่านค่าจาก ID หรือ Barcode ต่างๆ การมองเห็นของเครื่องจักรจำเป็นที่จะต้องมีการใช้งานเซนเซอร์ในการรับภาพนิ่ง (Image) หรือภาพเคลื่อนไหว (Video) ซึ่งเปรียบเสมือนกับดวงตาของหุ่นยนต์

ด้วยความยืดหยุ่นในการควบคุมหุ่นยนต์ DobotM1 ที่มี Python-API สำหรับการเขียนโปรแกรมภาษา Python สำหรับการควบคุมหุ่นยนต์ ทำให้สามารถนำเทคนิค Machine Vision มาใช้งานได้ โดยจะใช้งานไลบรารีที่ชื่อว่า OpenCV ซึ่งเป็นซอฟต์แวร์รหัสเปิด (Open Source) มีฟังก์ชันและคำสั่งต่างๆ จำนวนมากให้เลือกใช้งาน



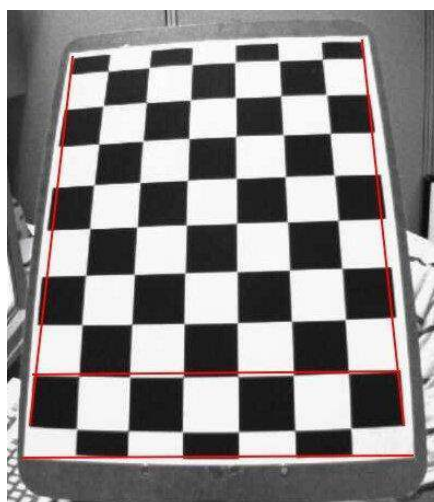
รูปที่ 30 การตรวจจับวัตถุจากสีโดยใช้ OpenCV

ในหัวข้อนี้จะเป็นการทำงานร่วมกันระหว่าง DobotAPI สำหรับการสั่งการทำงานของหุ่นยนต์ และ OpenCV ที่ใช้เทคนิคการประมวลผลภาพ (Image Processing) โดยเขียนโปรแกรมควบคุมผ่าน Visual Studio Code โดยสิ่งที่จะต้องมีการเตรียมการก่อนใช้งาน มีดังนี้

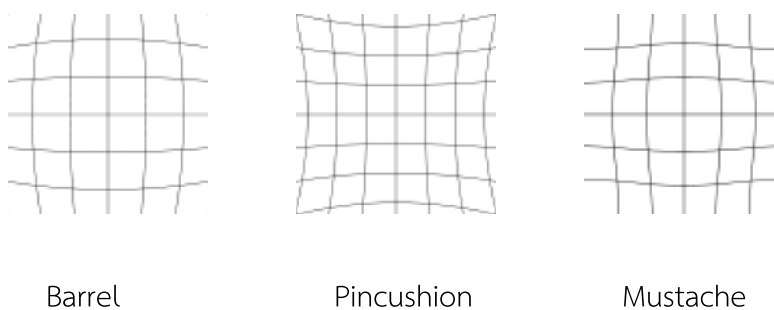
1. เตรียมการตามขั้นตอนการควบคุมหุ่นยนต์ผ่าน API
2. ติดตั้งแพ็คเกจ OpenCV บน Virtual Environment : `pip install opencv-python`
3. เชื่อมต่อหุ่นยนต์กับคอมพิวเตอร์ด้วย USB
4. เชื่อมต่อกล้อง (Webcam) กับคอมพิวเตอร์ด้วย USB
5. การสอบเทียบกล้อง
6. การสอบเทียบตำแหน่ง
7. การเขียนคำสั่งควบคุมหุ่นยนต์

การสอบเทียบกล้อง (Camera Calibration)

การสอบเทียบกล้องเป็นขั้นตอนที่สำคัญอย่างยิ่งต่อการใช้งานหุ่นยนต์ร่วมกับกล้องในการประมวลผลภาพ สำหรับการมองเห็นของเครื่องจักร (Machine Vision) เนื่องจากความนูนของเลนส์กล้องอาจส่งผลให้ภาพเกิดการบิดเบี้ยวของภาพได้ (Distortion) โดยในการถ่ายภาพจะเกิดการบิด 2 ประเภท ได้แก่ การบิดแบบโค้ง (Radial distortion) และการบิดแบบวงสัมผัส (Tangential distortion) โดยทำให้วัตถุภายในภาพที่เป็นเส้นตรงเกิดความโค้ง และเมื่อยิ่งห่างจากจุดศูนย์กลางของภาพการบิดของวัตถุจะยิ่งสูงขึ้น



รูปที่ 31 Radial distortion



รูปที่ 32 ลักษณะของ Radial distortion รูปแบบต่างๆ

สมการแสดงการบิดเบือน Radial Distortion ตามระยะความกว้าง (x) และความสูง (y) แสดงได้ดังนี้

$$x_{distorted} = x(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (1)$$

$$y_{distorted} = y(1 + k_1r^2 + k_2r^4 + k_3r^6) \quad (2)$$

สมการแสดงการบิดเบือน Tangential distortion ตามระยะความกว้าง (x) และความสูง (y) แสดงได้ดังนี้

$$x_{distorted} = x + [2p_1xy + p_2(r^2 + 2x^2)] \quad (3)$$

$$y_{distorted} = y + (p_1(r^2 + 2y^2) + 2p_2xy) \quad (4)$$

จากสมการที่ 1 – 4 จะพบว่า มีตัวแปรสำคัญจำนวน 5 ตัวแปรที่ต้องทราบ เรียกว่า Distortion coefficients

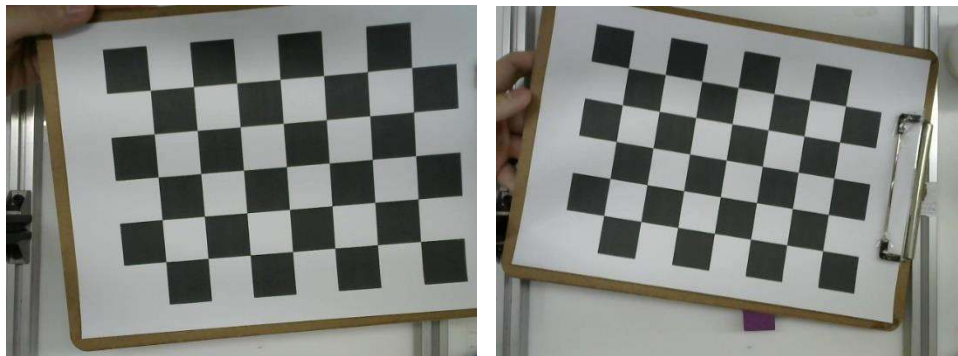
$$Distortion\ coefficients = [k_1\ k_2\ p_1\ p_2\ k_3] \quad (5)$$

นอกจากนี้ตัวแปรภายนอกที่จำเป็นต้องทราบคือ ความยาวโฟกัสของกล้อง (Focal length) ได้แก่ f_x และ f_y และระยะจุดกึ่งกลางเลนส์ (Optical center) ได้แก่ตัวแปร c_x และ c_y ตัวแปรดังกล่าวนี้จะถูกนำมาสร้าง Camera matrix ซึ่งเป็นเมทริกซ์ขนาด 3x3 เฉพาะของกล้องแต่ละตัวสำหรับใช้ในการแก้ไขความผิดเพี้ยนจากการบิดของภาพ แสดงได้ดังนี้

$$camera\ matrix = \begin{bmatrix} f_x & 0 & c_x \\ 0 & f_y & c_y \\ 0 & 0 & 1 \end{bmatrix} \quad (6)$$

ตัวแปรภายนอกดังกล่าวนี้ถูกใช้ในการคำนวณ Rotation Vector และ Translation Vector สำหรับแปลงค่าพิกัดในจุดสามมิติเป็นระบบพิกัดสองมิติ ในงานด้านการถ่ายภาพที่ต้องการทราบตำแหน่งที่แม่นยำจำเป็นอย่างยิ่งที่ต้องมีการแก้ไขความผิดเพี้ยนของภาพ อุปกรณ์ที่สำคัญที่ใช้ในการคำนวณหาตัวแปรที่ได้กล่าวมาข้างต้นคือ กระดานหมากรุก (Chess board) โดยใช้จุดตัดของตารางหมากรุกสีดำเป็นตำแหน่งอ้างอิง โดยทำการคำนวณระยะของพิกัดในจุดตัดจากความเป็นจริง ย้อนกลับไปหาระยะของพิกัดจุดตัดที่ได้จากภาพ จะทำให้ได้ตัวแปรสำคัญที่ใช้ในการแก้ไขความผิดเพี้ยนของภาพได้ โดยมีหลักการในการเก็บภาพดังนี้

1. เตรียมภาพกระดานหมากรุกที่ได้จากกล้องที่ต้องการสอบเทียบอย่างน้อย 20 ภาพ
2. ควรถ่ายภาพกระดานหมากรุกให้เห็นจุดตัดทุกจุดอย่างชัดเจน
3. ภาพกระดานหมากรุกที่ใช้ควรมีลักษณะเป็นแผ่นราบเรียบ
4. ควรมีการหมุนภาพหรือเอียงภาพในทุกทิศทาง และมีการเลื่อนระยะใกล้ไกลที่หลากหลาย เพื่อให้มีตัวอย่างที่เพียงพอต่อการคำนวณ และเพิ่มความแม่นยำของการสอบเทียบ



รูปที่ 33 ตัวอย่างของภาพที่ใช้ในการสอบเทียบ

การเตรียมข้อมูลภาพถ่ายสามารถทำการเก็บข้อมูลได้ โดยสร้างโฟลเดอร์สำหรับการเก็บข้อมูล เช่น images สำหรับ code ที่ใช้ในการเก็บข้อมูล มีดังนี้

```
[1] import cv2
[2] cap = cv2.VideoCapture(0)                                # Webcam กล้องหมายเลข 0
[3] num = 0
[4] while cap.isOpened():
[5]     succes, img = cap.read()
[6]     k = cv2.waitKey(5)
[7]     if k == 25:                                           # กำหนดจำนวนรูปถ่ายสูงสุด 25 รูป
[8]         break
[9]     elif k == ord('s'):                                    # กดปุ่ม s เพื่อกดถ่ายภาพและบันทึก
[10]         cv2.imwrite('ตำแหน่งไฟล์/images/img' + str(num) + '.png', img) # ใส่ตำแหน่งให้ครบ
[11]         print("image saved!")                             # แสดงข้อความ Image saved! เมื่อบันทึกภาพสำเร็จ
[12]         num += 1
[13]     cv2.imshow('Img',img)
[14] cap.release()
[15] cv2.destroyAllWindows()
```

Code นี้เป็นการเก็บภาพจากกล้อง Webcam โดยใช้กล้องหมายเลข 0 (ให้ตรวจสอบหมายเลขกล้องให้ถูกต้องโดยอาจเปลี่ยนหมายเลขเป็น 1, 2, 3... ทำการกดถ่ายภาพโดยกดปุ่ม s จะทำการบันทึกภาพถ่ายลงในโฟลเดอร์ images ชื่อไฟล์ imgX.png โดย X คือหมายเลขภาพเช่น img0.png img1.png ...

สำหรับขั้นตอนการสอบเทียบกล้องจะต้องมีการวัดขนาดของตารางสีดำบนกระดานหมากรุกในหน่วย mm ขนาดของรูป และจำนวนจุดตัดสีดำของตาราง code ที่ใช้มีดังนี้

```
[1] import numpy as np
[2] import cv2 as cv
[3] import glob
[4] import pickle

# หาตำแหน่งมุมของกระดานหมากรุก
[5] chessboardSize = (7,5)      # จำนวนจุดตัดของสี่เหลี่ยมสีดำ (ยาว,กว้าง)
[6] frameSize = (640,480)      # ขนาดของรูปที่ถ่ายมา
[7] # termination criteria
[8] criteria = (cv.TERM_CRITERIA_EPS + cv.TERM_CRITERIA_MAX_ITER, 30, 0.001)
[9] objp = np.zeros((chessboardSize[0] * chessboardSize[1], 3), np.float32)
[10] objp[:,2] = np.mgrid[0:chessboardSize[0],0:chessboardSize[1]].T.reshape(-1,2)
[11] size_of_chessboard_squares_mm = 28.50      # ขนาดของสี่เหลี่ยม mm
[12] objp = objp * size_of_chessboard_squares_mm
[13] objpoints = [ ] # 3d point in real world space
[14] imgpoints = [ ] # 2d points in image plane.
# แก๊บริกต์นี้ ให้เข้ามาเก็บไฟล์ใน folder images
[15] images = glob.glob('ตำแหน่งไฟล์ \images\*.png')
[16] for image in images:
[17]     img = cv.imread(image)
[18]     gray = cv.cvtColor(img, cv.COLOR_BGR2GRAY)

# หามุมของกระดานหมากรุก
[19]     ret, corners = cv.findChessboardCorners(gray, chessboardSize, None)
[20]     if ret == True:
[21]         objpoints.append(objp)
[22]         corners2 = cv.cornerSubPix(gray, corners, (11,11), (-1,-1), criteria)
[23]         imgpoints.append(corners)
```


วาดเส้นเชื่อมมุมของกระดานหมากรุก

```
[24] cv.drawChessboardCorners(img, chessboardSize, corners2, ret)
[25] cv.imshow('img', img)
[26] cv.waitKey(1000)
[27] cv.destroyAllWindows()
```

สอบเทียบ

```
[28] ret, cameraMatrix, dist, rvecs, tvecs = cv.calibrateCamera(objpoints, imgpoints,
    frameSize, None, None)
```

บันทึกผลการสอบเทียบ

```
[29] pickle.dump((cameraMatrix, dist), open( "calibration.pkl", "wb" ))
[30] pickle.dump(cameraMatrix, open( "cameraMatrix.pkl", "wb" ))
[31] pickle.dump(dist, open( "dist.pkl", "wb" ))
```

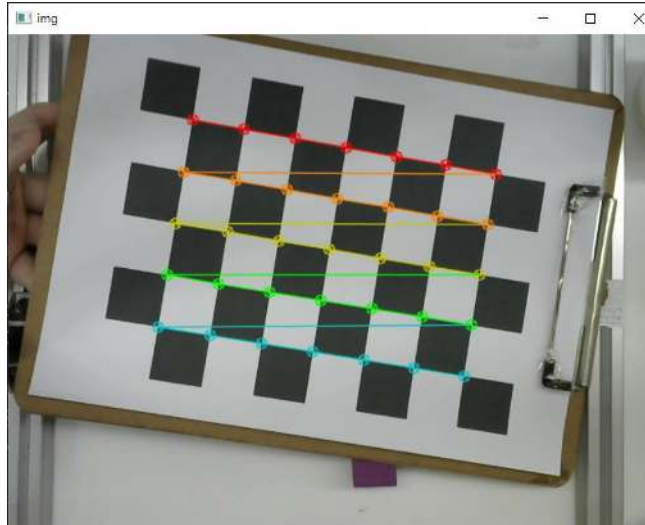
แก่การบิดเบี้ยวของภาพ

```
[32] img = cv.imread('ตำแหน่งไฟล์\images\img0.png') # แก่บรรทัดนี้
[33] h = 480
[34] w = 640
[35] newCameraMatrix, roi = cv.getOptimalNewCameraMatrix(cameraMatrix, dist, (w,h), 1,
    (w,h))
```

Print ค่า Matrix ที่ต้องการ

```
[36] print(dist)
[37] print(newCameraMatrix)
```

ผลการสอบเทียบจะได้ไฟล์ calibration.pkl, cameraMatrix.pkl และ dist.pkl และผลการลากเส้นเพื่อสอบเทียบภาพ แสดงดังรูป



รูปที่ 34 ผลการลากเส้นเชื่อมจุดตัดเพื่อทำการสอบเทียบ
(7 จุดในด้านยาวและ 5 จุดในด้านกว้าง)

สำหรับค่า Matrix ที่จะใช้ในการสอบเทียบภาพจะแสดงใน Terminal ดังรูป

```
images = glob.glob('D:\Desktop\MEX\MEX_Project_Robot\DobotM1\DobotM1_
d:\Desktop\MEX\MEX_Project_Robot\DobotM1\DobotM1_Python\My Camera Calib
img = cv.imread('D:\Desktop\MEX\MEX_Project_Robot\DobotM1\DobotM1_Pyt
[[-2.05652075e-02  8.94169355e-01 -4.18366300e-03  1.55534521e-03
 -2.67575819e+00]]
[[782.10754739   0.          305.12730107]
 [  0.          783.2372439  247.15385808]
 [  0.           0.           1.          ]]
PS D:\Desktop\MEX\MEX_Project_Robot\DobotM1\DobotM1_Python>
```

รูปที่ 35 ผลการหา distortion Coefficients และ camera matrix

ในการแก้การบิดเบี้ยวของภาพจะใช้ distortion coefficient และ camera matrix โดยการตั้งชื่อตัวแปรแบบอาเรย์ สามารถแสดงได้ดังนี้

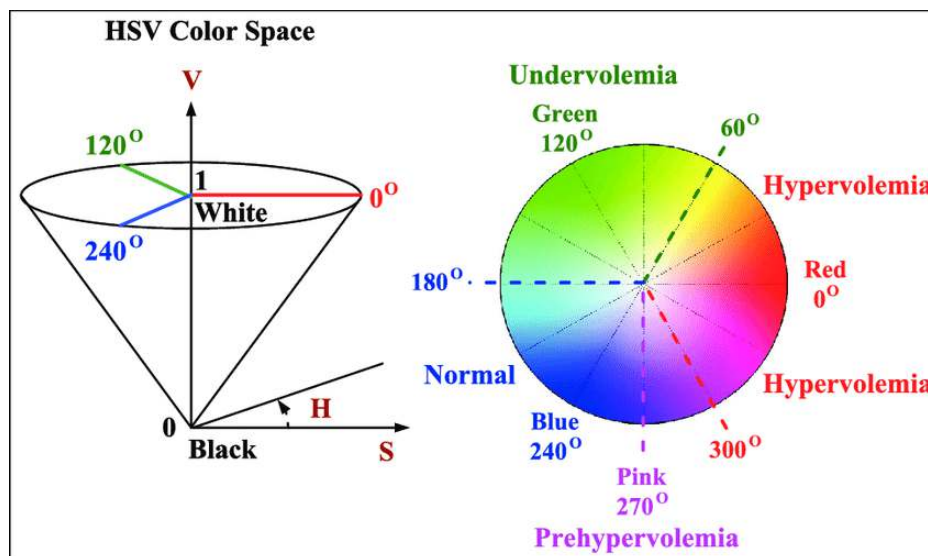
```
[1] dist = np.array([[ -5.17994872e-02, 1.33860825e+00, -2.93913838e-03, -3.28111848e-03, -  
5.27297925e+00]])  
[2] cameraMatrix = np.array([[816.53414246, 0., 284.80505879],  
[0., 809.129015, 246.80347752],  
[0., 0., 1.]])
```

ตัวอย่าง code สำหรับการแก้การบิดเบี้ยวของภาพที่ได้จากกล้อง webcam แสดงได้ดังนี้

```
[1] import cv2  
[2] import numpy as np  
[3] cap = cv2.VideoCapture(0)  
[4] dist = np.array([[ -5.17994872e-02, 1.33860825e+00, -2.93913838e-03, -3.28111848e-03, -  
5.27297925e+00]])  
[5] cameraMatrix = np.array([[816.53414246, 0., 284.80505879],[0., 809.129015,  
246.80347752], [0., 0., 1.]])  
[6] while True:  
[7]     check, frame = cap.read()  
    # แก้ Distorsion  
[8]     newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist,  
        (640,480), 1, (640,480))  
[9]     dst_frame = cv2.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)  
[10]    cv2.imshow("Undistorsion VDO", dst_frame)  
[11]    if cv2.waitKey(1) & 0xFF == ord("e"):  
[12]        break  
[13] cap.release()  
[14] cv2.destroyAllWindows()
```

การตรวจจับวัตถุจากสี (Color Recognition)

สำหรับการตรวจจับวัตถุจากสีจะใช้ไลบรารี OpenCV โดยเริ่มจากการแปลงรูปหรือวิดีโอจากระบบสี BGR เป็นระบบสี HSV จากนั้นกำหนดช่วงสีต่ำสุด (lower) และช่วงสีสูงสุด (Upper) ของวัตถุที่ต้องการ การกำหนดค่าสีอาจดูได้จากวงกลม และกรวยสีในระบบ HSV ดังรูป



รูปที่ 36 วิธีการอ่านค่าสีในระบบ HSV

(https://www.researchgate.net/figure/HSV-color-space-and-RGB-color-transformation_fig4_312678134)

- การอ่านค่า Hue (H) อ่านได้จากวงกลมสีโดยมีค่าระหว่าง 0 – 359 จากนั้นทำการหารค่าที่อ่านได้ด้วย 2 เนื่องจาก OpenCV กำหนดค่าระหว่าง 0 – 179
- การอ่านค่า Saturation (S) อ่านจากเส้นผ่านศูนย์กลางของวงกลมถึงขอบของวงกลม OpenCV กำหนดค่าระหว่าง 0 – 255
- การอ่านค่า Value (V) อ่านจากด้านบนลงล่างของกรวย OpenCV กำหนดค่าระหว่าง 0 – 255

ตัวอย่าง Code:

```
[1] cap = cv2.VideoCapture(0)
[2] dist = np.array([[[-5.17994872e-02, 1.33860825e+00, -2.93913838e-03, -3.28111848e-03, -
    5.27297925e+00]])
[3] cameraMatrix = np.array([[816.53414246, 0., 284.80505879],
    i. [0., 809.129015, 246.80347752],
    ii. [0., 0., 1.]])
[4] while True:
[5]     cmd = input("Enter command: ")          # รับค่าจาก Keyboard
[6]     if cmd == "x":                          # รับค่าปุ่ม x ให้เริ่มการทำงาน
[7]         check, frame = cap.read()
# ทำการแก้การบิดเบี้ยวของภาพ
[8]         newCameraMatrix, roi = cv2.getOptimalNewCameraMatrix(cameraMatrix, dist,
    (640,480), 1, (640,480))
[9]         dst_frame = cv2.undistort(frame, cameraMatrix, dist, None, newCameraMatrix)
# การตรวจจับวัตถุจากสี ตัวอย่าง สีเขียว (Green)
[10]        hsv = cv2.cvtColor(dst_frame, cv2.COLOR_BGR2HSV) # แปลงระบบสี BGR เป็น HSV
[11]        lower_green = np.array([40, 120, 80])          # ค่าสีต่ำสุด
[12]        upper_green = np.array([85, 255, 255])         # ค่าสีสูงสุด
[13]        mask_green = cv2.inRange(hsv, lower_green, upper_green)
# ปรับภาพโดยใช้การ Dilation เพื่อให้สามารถสร้างเส้นเค้าโครงขอบภาพได้ง่ายยิ่งขึ้น
[14]        kernel = np.ones((5, 5), np.uint8)
[15]        dilated = cv2.dilate(mask_green, kernel, iterations=3) # ปรับค่า Iteration ได้
[16]        cv2.imshow("dilated", dilated) # สำหรับตรวจสอบภาพหลังการทำ dilation
# สร้างเส้นเค้าโครง (contour) รอบวัตถุ
[17]        contours_green, _ = cv2.findContours(dilated, cv2.RETR_TREE,
    cv2.CHAIN_APPROX_SIMPLE)
```

สร้าง loop ในการตรวจจับวัตถุหลายชิ้น

```
[18] for cnt in contours_green:
```

```
[19]     contour_area = cv2.contourArea(cnt)
```

```
[20]     if contour_area > 1000:
```

```
[21]         x, y, w, h = cv2.boundingRect(cnt)
```

สร้างกรอบสี่เหลี่ยมสีเขียวบนตำแหน่งซ้ายบน-ขวาล่างครอบวัตถุที่ตรวจจับได้

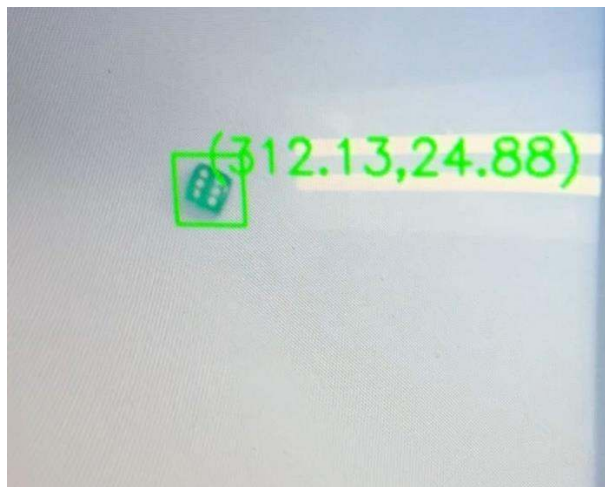
```
[22]         cv2.rectangle(cap, (x, y), (x + w, y + h), (0, 255, 0), 2)
```

```
[23]         cv2.putText(cap, 'Green', (x, y-10), cv2.FONT_HERSHEY_SIMPLEX, 0.9, (0, 255, 0), 2)
```

```
[24] cv2.imshow('Color Recognition Output', cap)
```

```
[25] if cv2.waitKey(1) & 0xFF == ord('e'):
```

```
[26]     break
```



รูปที่ 37 ผลลัพธ์การตรวจจับวัตถุจากสี

ตัวอย่างการใช้งาน Machine Vision กับหุ่นยนต์ Dobot M1

ในหัวข้อนี้เป็นการรวมองค์ความรู้ทั้งหมดจากหัวข้อการควบคุมหุ่นยนต์ผ่าน Python API การสอบเทียบกล้อง และการตรวจจับวัตถุจากสี โดยตัวอย่างนี้เป็นการแสดงวิธีการตรวจจับลูกเต๋าหลายสีและหลายชิ้น เพื่อให้หุ่นยนต์ DobotM1 ทำการหยิบลูกเต๋าล้วนนำไปวางใส่ถ้วยหรือถาดแยกแต่ละสีที่มีการกำหนดตำแหน่งของถ้วยไว้คงที่สามารถสรุปขั้นตอนเพื่อให้เข้าใจภาพรวมได้ดังนี้

- 1) สอบเทียบกล้อง (Camera Calibration) เพื่อให้ได้ camera matrix และ distortion coefficients
- 2) แก้การบิดเบี้ยวของภาพเพื่อให้ได้ภาพที่มีการแก้ไขการบิดเบี้ยวแล้ว (Undistorted Image/VDO)
- 3) ทำการกำหนดช่วงสีในระบบ HSV สร้างหน้ากาก (mask), เส้นเค้าโครง (contour) ตีกรอบ (Bounding box) รอบวัตถุที่ตรวจจับได้โดยใช้ for loop ตรวจจับวัตถุหลายชิ้น
- 4) สอบเทียบตำแหน่งของวัตถุนบนพื้นที่ทำงาน (Workspace) กับตำแหน่งของกล้อง และตำแหน่งของหุ่นยนต์ เพื่อให้ได้ตำแหน่งที่สั่งให้หุ่นยนต์หยิบชิ้นงาน
- 5) ใช้คำสั่งให้หุ่นยนต์หยิบวัตถุที่ตำแหน่งจุดศูนย์กลาง (Centroid) ของวัตถุ และปล่อยวัตถุที่ตำแหน่งที่กำหนดไว้แบบแยกตามลำดับสี

ขั้นตอนในเบื้องต้นมีเพียงเท่านี้โดยสามารถเพิ่มความซับซ้อนของขั้นตอน ได้ดังตัวอย่างต่อไปนี้

- หยิบวัตถุสีเดียวกันหลายชิ้น นำมาจับวางเป็นแถวแนวเส้นตรง
- หยิบวัตถุหลายสีแยกเป็นกอง โดยจะหยิบวัตถุสีใดก่อนก็ได้
- หยิบวัตถุหลายสีแยกเป็นกอง โดยเลือกหยิบทีละสี
- เลือกหยิบวัตถุเฉพาะสีที่ต้องการ จากวัตถุหลายสี

ตัวอย่างการทำงานที่ต้องการแสดงได้ดัง YouTube Shorts :

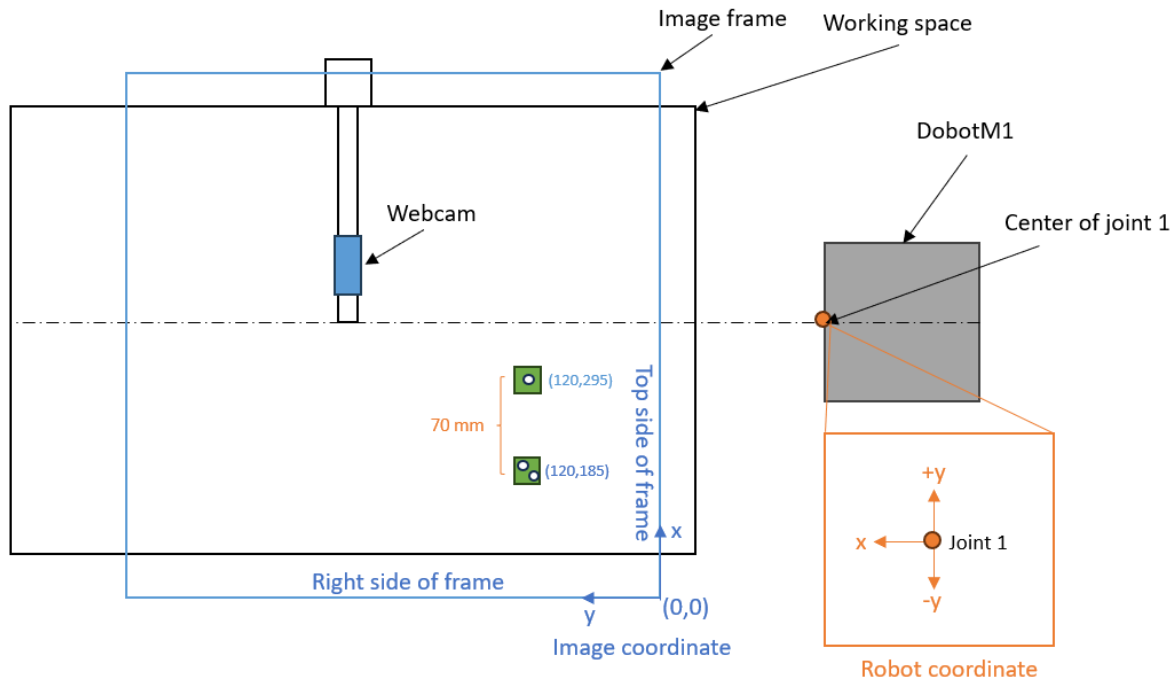
- ตรวจจับและหยิบลูกเต๋าสีเขียว

<https://youtube.com/shorts/Yr4ZtG9e2nE?feature=share>

- ตรวจจับและหยิบลูกสองสี

<https://youtube.com/shorts/B1hMO9uchBc?feature=share>

สำหรับตัวอย่างนี้จะแสดงวิธีการตรวจจับวัตถุ 2 สี คือ ลูกเต๋าสีเขียว และลูกเต๋าสีม่วง โดยทำการสร้างพื้นที่ทำงานที่มีการติดตั้งกล้อง webcam อยู่บนฐาน ดังรูป



รูปที่ 38 ลักษณะของพื้นที่ทำงานและการติดตั้งกล้อง

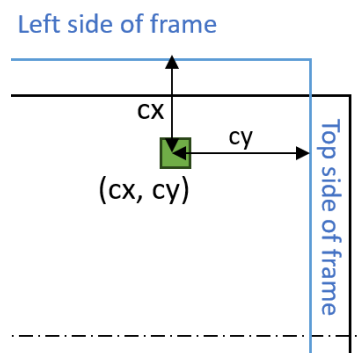
จากรูปจะพบว่าขอบเขตของภาพ (Image frame) จะมีส่วนซ้อนและเกินออกจากพื้นที่ทำงาน (Working space) นอกจากนี้ยังมีตำแหน่งของระบบพิกัดฉาก (Coordinate) ที่แตกต่างกัน ดังนั้นจึงมีความจำเป็นอย่างยิ่งที่ต้องมีการสอบเทียบระยะของภาพ และพื้นที่ทำงาน เพื่อให้รู้ระยะในการควบคุมหุ่นยนต์ได้ โดยมีขั้นตอน ดังนี้

- 1) คำนวณหาขนาดของระยะจริง (mm) กับระยะภายในภาพ (pixel หรือ px) เป็นตัวแปร mm_per_px
- 2) คำนวณตำแหน่งของวัตถุภายในภาพให้เป็นหน่วย mm
- 3) คำนวณตำแหน่งของ X และ Y ที่ต้องการให้หุ่นยนต์ทำการหยิบชิ้นงาน

วิธีการสอบเทียบระยะ

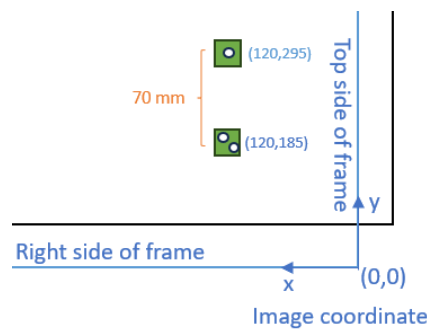
ขั้นตอนที่ 1 วางวัตถุสองชิ้นเพื่อให้กล้องตรวจจับโดยใช้วิธีการตรวจจับวัตถุจากสี อ่านค่าตำแหน่งภายในภาพคือ ระยะ x , y , w และ h ให้คำนวณหาระยะถึงจุดศูนย์กลางของวัตถุ (Centroid) หรือ c ได้ระยะ cx และ cy ของตำแหน่งวัตถุ มีหน่วยเป็น Pixel (px) คำนวณได้ ดังนี้

$$cx = x + (w/2) \quad \text{และ} \quad cy = y + (h/2)$$



รูปที่ 39 การระบุตำแหน่งของวัตถุบนภาพ (cx,cy)

อ่านค่าพิกัดที่ตรวจจับได้ (สีฟ้า) และใช้ไม้บรรทัดวัดค่าระยะห่างระหว่างวัตถุจริงในหน่วย mm (สีส้ม) ดังรูป



รูปที่ 40 การหาค่า mm per pixel

คำนวณ mm_per_px ได้ดังนี้

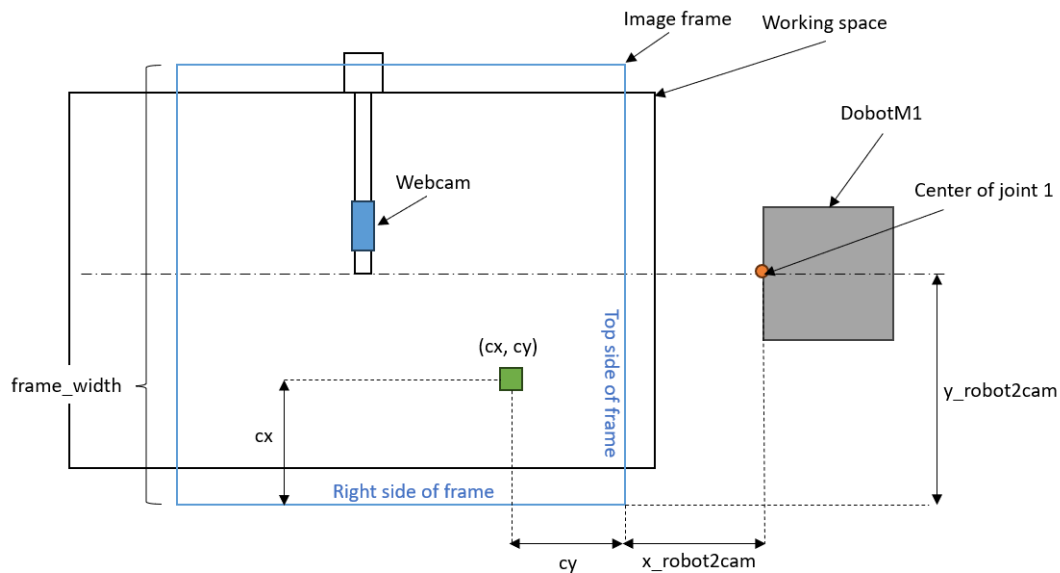
$$\text{mm_per_px} = (70) / (295 - 185) = 0.6364$$

ขั้นตอนที่ 2 แปลงหน่วยของตำแหน่งที่ได้จากภาพเป็นหน่วย mm ตั้งชื่อตัวแปรว่า mm_cx_offset และ mm_cy_offset คำนวณได้ดังนี้

$$\text{mm_cx_offset} = (\text{cx}) (\text{mm_per_px})$$

$$\text{mm_cy_offset} = (\text{cy}) (\text{mm_per_px})$$

ขั้นตอนที่ 3 คำนวณตำแหน่ง x และ y ของชิ้นงานให้หุ่นยนต์หยิบจับชิ้นงาน โดยมีระยะต่างๆ ดังรูป



รูปที่ 41 การวัดระยะเพื่อทำการสอบเทียบระยะ

ระยะระหว่าง Joint 1 ของหุ่นยนต์ถึงขอบภาพ (Image frame) แทนด้วยตัวแปร x_robot2cam และ y_robot2cam ตัวอย่างอ่านค่าระยะได้ดังนี้

$$x_robot2cam = 206 \text{ mm}$$

$$y_robot2cam = 219.5 \text{ mm}$$

สร้างสมการให้หุ่นยนต์หับชิ้นงานบนแกน X ได้ดังนี้

$$x_real = mm_cy_offset + x_robot2cam$$

สร้างสมการให้หุ่นยนต์หับชิ้นงานบนแกน Y ได้ดังนี้

$$y_real = y_robot2cam - (frame_width - cx) (mm_per_px)$$

ทั้งนี้จะเขียนแทน $(frame_width - cx) (mm_per_px)$ ด้วยตัวแปร y_2

$$y_2 = (frame_width - cx) (mm_per_px)$$

นอกจากนี้ในการกำหนดตำแหน่งอาจมีความผิดพลาดเกิดขึ้น จะมีการบวกค่าคงที่เพื่อปรับระยะโดยจะกำหนดค่าเริ่มต้นเป็น 0 ก่อน เมื่อพบว่ามีความคลาดเคลื่อนในแกนใดจะให้ทำการบวกหรือลบค่าตามความคลาดเคลื่อนที่เกิดขึ้นในแกนนั้น ๆ ใช้ตัวแปร x_trim และ y_trim จะได้สมการดังนี้

$$x_real = mm_cy_offset + x_robot2cam + x_trim$$

$$y_real = y_robot2cam - y_2 + y_trim$$

เมื่อได้สมการสอบเทียบระยะทั้งหมดแล้วก็จะสามารถทำการหับวัตถุได้โดยการระบุตำแหน่งด้วยการตรวจจับวัตถุจากสี ดังตัวอย่างถัดไป