client.py

```python
import cv2
import json
import os
import numpy as np

# ========= Default CONFIG (used on first run or after Reset) =========
DEFAULT_HSV_RANGES = [
    {"name": "Yellow", "lower": (3,   137, 131), "upper": (47,  226, 220)},
    {"name": "Red",    "lower": (150, 147, 137), "upper": (226, 255, 196)},
    {"name": "Blue",   "lower": (87,   77,  63), "upper": (150, 250, 255)},
    {"name": "Green",  "lower": (36,   68, 114), "upper": (74,  219, 184)},
]

# Per-range drawing colors (B,G,R). Extend/cycle as needed.
DRAW_COLORS = [(0,255,255), (0,0,255), (255,0,0), (0,255,0)]

# Contour filtering & drawing
MIN_AREA  = 500     # ignore tiny blobs
BOX_TYPE  = 0       # 0 = axis-aligned, 1 = rotated
THICKNESS = 2

# Save file
CONFIG_PATH = "hsv_config.json"
```

กำหนดค่าเริ่มต้นต่าง ๆ ของโปรแกรม

DEFAULT_HSV_RANGES: กำหนดช่วงค่าสี HSV เริ่มต้นสำหรับสีต่างๆ (เหลือง, แดง, น้ำเงิน, เขียว)

DRAW_COLORS: กำหนดสีสำหรับวาดกรอบรอบวัตถุที่ตรวจจับได้ (ใช้สี BGR)

MIN_AREA, BOX_TYPE, THICKNESS: พารามิเตอร์สำหรับกรองวัตถุขนาดเล็ก, ประเภทของกรอบ และความหนาของเส้น

CONFIG_PATH: ชื่อไฟล์สำหรับบันทึกการตั้งค่า (hsv_config.json)

```python
def compute_distortion_map(w, h, k):
    """Simple radial barrel/pincushion remap. k in [-1,+1]."""
    cx, cy = (w - 1) / 2.0, (h - 1) / 2.0
    max_rad = np.sqrt(cx**2 + cy**2)
    x = np.linspace(0, w - 1, w, dtype=np.float32)
    y = np.linspace(0, h - 1, h, dtype=np.float32)
    X, Y = np.meshgrid(x, y)
    Xc, Yc = X - cx, Y - cy
    xn, yn = Xc / max_rad, Yc / max_rad
    r2 = xn*xn + yn*yn
    scale = 1.0 + k * r2
    Xd = (xn * scale) * max_rad + cx
    Yd = (yn * scale) * max_rad + cy
    return Xd.astype(np.float32), Yd.astype(np.float32)

def ensure_odd(n): return max(1, n | 1)
def nothing(_): pass
```

ฟังก์ชันนี้ใช้แก้ไขความบิดเบี้ยวของภาพที่เกิดจากเลนส์กล้อง (ภาพป่องกลางหรือเว้ากลาง) โดยจะสร้าง map สำหรับ cv2.remap เพื่อปรับแก้ภาพให้ตรงก่อนนำไปประมวลผลต่อ

```python
# ================ Load/Save config ====================
def load_config_or_default():
    if os.path.exists(CONFIG_PATH):
        with open(CONFIG_PATH, "r", encoding="utf-8") as f:
            data = json.load(f)
        return {
            "ranges": data.get("ranges", DEFAULT_HSV_RANGES),
            "distortion_kx100": int(data.get("distortion_kx100", 100)),
            "morph_op": int(data.get("morph_op", 0)),
            "kernel_size": int(data.get("kernel_size", 3)),
            "iterations": int(data.get("iterations", 1)),
        }
    else:
        return {
            "ranges": DEFAULT_HSV_RANGES,
            "distortion_kx100": 100,
            "morph_op": 0,
            "kernel_size": 3,
            "iterations": 1,
        }

def save_config(ranges, kx100, morph_op, ksize, iters):
    data = {
        "ranges": ranges,  # [{"name":..., "lower":[H,S,V], "upper":[H,S,V]}, ...]
        "distortion_kx100": int(kx100),
        "morph_op": int(morph_op),
        "kernel_size": int(ksize),
        "iterations": int(iters),
    }
    # convert tuples to lists (json friendly)
    for r in data["ranges"]:
        r["lower"] = list(r["lower"])
        r["upper"] = list(r["upper"])
    with open(CONFIG_PATH, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2)
    print(f"Saved settings to {CONFIG_PATH}")
```

ทำหน้าที่บันทึกและโหลดการตั้งค่าต่างๆ เพื่อให้ไม่ต้องตั้งค่าใหม่ทุกครั้งที่เปิดโปรแกรม

load_config_or_default(): ตรวจสอบว่ามีไฟล์ hsv_config.json หรือไม่ ถ้ามีจะโหลดค่าที่เคยบันทึกไว้มาใช้ ถ้าไม่มีจะใช้ค่าเริ่มต้นจาก DEFAULT_HSV_RANGES

save_config(): บันทึกค่าที่ปรับจากหน้าจอ UI (เช่น ค่า HSV, ค่า Distortion) ลงในไฟล์ hsv_config.json ในรูปแบบ JSON

```python
# ================= UI creation helpers ================
def create_controls_windows(cfg):
    # Global controls
    cv2.namedWindow("Controls", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Controls", 520, 400)
    cv2.createTrackbar("Distortion k x100", "Controls", cfg["distortion_kx100"], 200, nothing)
    cv2.createTrackbar("Morph Op (0-4)", "Controls", cfg["morph_op"], 4, nothing)
    cv2.createTrackbar("Kernel Size", "Controls", cfg["kernel_size"], 31, nothing)
    cv2.createTrackbar("Iterations", "Controls", cfg["iterations"], 10, nothing)

    # Add trackbars for world coordinates for up to 4 cubes
    for i in range(4):
        cv2.createTrackbar(f"Cube{i+1} World X", "Controls", 0, 1000, nothing)
        cv2.createTrackbar(f"Cube{i+1} World Y", "Controls", 0, 1000, nothing)

    # Per-range HSV windows
    for spec in cfg["ranges"]:
        win = f"HSV - {spec['name']}"
        cv2.namedWindow(win, cv2.WINDOW_NORMAL)
        cv2.resizeWindow(win, 420, 260)
        lh, ls, lv = spec["lower"]
        hh, hs, hv = spec["upper"]
        cv2.createTrackbar("Low H",  win, int(lh), 255, nothing)
        cv2.createTrackbar("Low S",  win, int(ls), 255, nothing)
        cv2.createTrackbar("Low V",  win, int(lv), 255, nothing)
        cv2.createTrackbar("High H", win, int(hh), 255, nothing)
        cv2.createTrackbar("High S", win, int(hs), 255, nothing)
        cv2.createTrackbar("High V", win, int(hv), 255, nothing)
```

```python
def read_controls(cfg):
    # Read global controls
    kx100 = cv2.getTrackbarPos("Distortion k x100", "Controls")
    morph = cv2.getTrackbarPos("Morph Op (0-4)", "Controls")
    ksize = ensure_odd(cv2.getTrackbarPos("Kernel Size", "Controls"))
    iters = cv2.getTrackbarPos("Iterations", "Controls")

    # Read per-range HSV
    ranges = []
    for spec in cfg["ranges"]:
        win = f"HSV - {spec['name']}"
        lh = cv2.getTrackbarPos("Low H",  win)
        ls = cv2.getTrackbarPos("Low S",  win)
        lv = cv2.getTrackbarPos("Low V",  win)
        hh = cv2.getTrackbarPos("High H", win)
        hs = cv2.getTrackbarPos("High S", win)
        hv = cv2.getTrackbarPos("High V", win)
        ranges.append({"name": spec["name"], "lower": (lh, ls, lv), "upper": (hh, hs, hv)})

    return ranges, kx100, morph, ksize, iters
```

สร้างหน้าต่างและแถบเลื่อน (Trackbar) สำหรับให้ผู้ใช้ปรับค่าต่างๆ ได้แบบ Real-time

create_controls_windows(): สร้างหน้าต่างควบคุมหลักชื่อ "Controls" สำหรับปรับค่า Distortion, Morphological Operations และสร้างหน้าต่างย่อยสำหรับปรับค่า HSV ของแต่ละสี

read_controls(): อ่านค่าปัจจุบันจาก Trackbar ทั้งหมดที่ผู้ใช้ปรับ เพื่อนำไปใช้ในการประมวลผลภาพในรอบถัดไป

```python
# ============== Contour selection (ONE per label) ==============
def score_contour(cnt):
    """Return a confidence score for a contour."""
    area = cv2.contourArea(cnt)
    if area <= 0:
        return 0.0
    x, y, w, h = cv2.boundingRect(cnt)
    box_area = max(1, w * h)
    fill_ratio = float(area) / float(box_area)  # 0..1
    return area * fill_ratio  # bigger & more compact blobs score higher

def pick_best_contour(contours, min_area=0):
    """Return (best_contour, score) or (None, 0)."""
    best = None
    best_score = 0.0
    for c in contours:
        if cv2.contourArea(c) < min_area:
            continue
        s = score_contour(c)
        if s > best_score:
            best = c
            best_score = s
    return best, best_score
```

หลังจากที่โปรแกรมสร้าง Mask ของสีที่สนใจแล้ว จะใช้ฟังก์ชันเหล่านี้เพื่อหาวัตถุที่ดีที่สุด

score_contour(): ให้คะแนนวัตถุ (Contour) แต่ละชิ้น โดยพิจารณาจาก ขนาด (Area) และ ความหนาแน่น (Fill Ratio) วัตถุที่ใหญ่และมีรูปทรงตันๆ (ไม่เว้าแหว่ง) จะได้คะแนนสูง

pick_best_contour(): วนลูปผ่าน Contour ทั้งหมดที่เจอใน Mask แล้วใช้ score_contour() เพื่อเลือก Contour ที่มีคะแนนดีที่สุดเพียงชิ้นเดียวสำหรับสีนั้นๆ วิธีนี้ช่วยป้องกันการตรวจจับวัตถุเดียวกันซ้ำซ้อนหรือตรวจเจอ Noise ที่ไม่ต้องการ

```python
def to_pos_robot(box1):
    camera_x, camera_y, r = box1[0], box1[1], box1[2]
    camera_coord = np.float32([[camera_x, camera_y]])
    robot_coord = cv2.perspectiveTransform(camera_coord.reshape(-1, 1, 2), matrix)
    robotx, roboty = robot_coord[0][0][0], robot_coord[0][0][1]
    return robotx, roboty, 90 - r

while True:
    ok, frame = cap.read()
    if not ok:
        continue

    h, w = frame.shape[:2]

    # Read UI
    ranges, kx100, morph, ksize, iters = read_controls(cfg)
    k_val = (kx100 - 100) / 100.0

    # Recompute remap if needed
    if (w != last_w) or (h != last_h) or (k_val != last_k) or (map_x is None):
        map_x, map_y = compute_distortion_map(w, h, k_val)
        last_w, last_h, last_k = w, h, k_val

    # Apply distortion ONCE
    distorted = cv2.remap(frame, map_x, map_y, interpolation=cv2.INTER_LINEAR, borderMode=

    hsv = cv2.cvtColor(distorted, cv2.COLOR_BGR2HSV)
    annotated = distorted.copy()
    all_masks = []

    # Build shared kernel (if any)
    kernel = None
    if ksize > 1 and iters > 0 and morph > 0:
        kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (ksize, ksize))

    cube_centroids = []
```

โมดูลนี้ใช้แปลงพิกัด

to_pos_robot(): ทำหน้าที่แปลงพิกัดของวัตถุจาก พิกัดของกล้อง (หน่วยเป็น pixel) ไปเป็น พิกัดของหุ่นยนต์ (หน่วยเป็น mm หรือ cm) โดยใช้ matrix ที่ได้จากการทำ Perspective Transform

Perspective Transform: ในโค้ดจะมีการกำหนด camera_points (จุดบนภาพจากกล้อง) และ world_points (จุดเดียวกันในโลกจริงที่วัดไว้) จากนั้นใช้ cv2.getPerspectiveTransform() เพื่อสร้าง matrix สำหรับการแปลงค่า ซึ่งผู้ใช้สามารถกด 'p' เพื่อให้โปรแกรมพิมพ์ค่า camera_points ของวัตถุที่ตรวจจับได้ออกมา เพื่อนำไปใช้คำนวณหา matrix นี้

```python
# ============================ Main ============================
def main():
    cfg = load_config_or_default()
    create_controls_windows(cfg)

    cap = cv2.VideoCapture(0,cv2.CAP_DSHOW) if hasattr(cv2, "CAP_DSHOW") else cv2.VideoCapture
    if not cap.isOpened():
        print("Error: Could not open camera 0")
        return

    last_w = last_h = -1
    last_k = None
    map_x = map_y = None
    font = cv2.FONT_HERSHEY_SIMPLEX

    print("Controls: S = save, R = reset to defaults P = ShowPOSITION, Q/ESC = quit")
    print("Press ] to rotate mask to each color, [ to show all masks.")

    mask_mode = "all"  # "all" or "single"
    mask_index = 0

    # --- Add your fixed calibration points and matrix ---
    camera_points = np.float32([[226, 84], [418, 164], [493, 314], [154, 372]])
    world_points = np.float32([[253.22, -39.64], [288.03, 38.95], [348.7, 70.22], [371.82, -70
    matrix = cv2.getPerspectiveTransform(camera_points, world_points)
    print("Perspective matrix:\n", matrix)
```

```python
# Process each HSV range
for i, spec in enumerate(ranges):
    lower = np.array(spec["lower"], dtype=np.uint8)
    upper = np.array(spec["upper"], dtype=np.uint8)
    mask = cv2.inRange(hsv, lower, upper)

    if kernel is not None:
        if morph == 1:   mask = cv2.erode(mask, kernel, iterations=iters)
        elif morph == 2: mask = cv2.dilate(mask, kernel, iterations=iters)
        elif morph == 3: mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kernel, iterati
        elif morph == 4: mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kernel, iterat

    all_masks.append(mask)

    # Find contours and keep ONLY ONE best contour
    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
    best_cnt, best_score = pick_best_contour(contours, MIN_AREA)
    color = DRAW_COLORS[i % len(DRAW_COLORS)]
    label_name = spec["name"]

    if best_cnt is not None:
        # centroid
        M = cv2.moments(best_cnt)
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
        else:
            x, y, w0, h0 = cv2.boundingRect(best_cnt)
            cx, cy = x + w0 // 2, y + h0 // 2

        cube_centroids.append([cx, cy])
```

```python
# bounding box (axis or rotated)
if BOX_TYPE == 0:
    x, y, w0, h0 = cv2.boundingRect(best_cnt)
    cv2.rectangle(annotated, (x, y), (x + w0, y + h0), color, THICKNESS)
else:
    rect = cv2.minAreaRect(best_cnt)
    box = np.int32(cv2.boxPoints(rect))
    cv2.polylines(annotated, [box], True, color, THICKNESS)

# draw centroid + text (name + coords + score)
cv2.circle(annotated, (cx, cy), 4, (0,0,0), -1)
cv2.circle(annotated, (cx, cy), 3, color, -1)
conf = f"{best_score:.0f}"

# Use robot coordinates for annotation
robotx, roboty, robotr = to_pos_robot([cx, cy, 0])
text = f"{label_name} Robot({robotx:.1f},{roboty:.1f}) conf:{conf}"
cv2.putText(annotated, text, (cx + 6, cy - 6), font, 0.6, (0,0,0), 3, cv2.LINE_
cv2.putText(annotated, text, (cx + 6, cy - 6), font, 0.6, (255,255,255), 1, cv2
```

```python
    key = cv2.waitKey(1) & 0xFF
    if key == ord(']'):
        mask_mode = "single"
        mask_index = (mask_index + 1) % len(all_masks)
    elif key == ord('['):
        mask_mode = "all"
    elif key == ord('p'):
        # Print out camera points in requested format
        if cube_centroids:
            print("camera_points = np.float32([", end="")
            print(", ".join(f"[{cx}, {cy}]" for cx, cy in cube_centroids), end="")
            print("])")
            # Print color order line
            color_order = " -> ".join([spec["name"] for spec in ranges])
            print(f"#{color_order}")
        else:
            print("No cubes detected.")

    if mask_mode == "all":
        combined_mask = np.zeros_like(all_masks[0]) if all_masks else np.zeros((h, w), np
        for m in all_masks:
            combined_mask = cv2.bitwise_or(combined_mask, m)
        cv2.imshow("Combined Mask", combined_mask)
    elif mask_mode == "single":
        if all_masks:
            cv2.imshow("Combined Mask", all_masks[mask_index])
        else:
            cv2.imshow("Combined Mask", np.zeros((h, w), np.uint8))

    cv2.imshow("Annotated Output", annotated)

    key = cv2.waitKey(1) & 0xFF
    if key in (27, ord('q')):
        save_config(ranges, kx100, morph, ksize, iters)
        break
    elif key in (ord('s'), ord('S')):
        save_config(ranges, kx100, morph, ksize, iters)
    elif key in (ord('r'), ord('R')):
        # Reload saved config from hsv_config.json (do NOT force DEFAULT_HSV_RANGES)
        print("Reloading saved config.")
        cv2.destroyAllWindows()
        cfg = load_config_or_default()
        create_controls_windows(cfg)

cap.release()
cv2.destroyAllWindows()
```

main() คือฟังก์ชันที่ควบคุมการทำงานทั้งหมดของโปรแกรม มีขั้นตอนดังนี้:

โหลดค่า Config และสร้างหน้าต่าง UI

เปิดกล้อง

เข้าสู่ Loop การทำงานที่ไม่สิ้นสุด

อ่านภาพจากกล้อง

อ่านค่าจาก UI Controls

ใช้ cv2.remap เพื่อแก้ไขความบิดเบี้ยวของภาพ

แปลงภาพจาก BGR เป็น HSV

วนลูปตามจำนวนสีที่ตั้งค่าไว้:

สร้าง Mask จากช่วงค่าสี HSV

ใช้ Morphological Operations (เช่น Erode, Dilate) เพื่อกำจัด Noise ใน Mask

ค้นหา Contours ทั้งหมดใน Mask

ใช้ pick_best_contour เพื่อเลือกวัตถุที่ดีที่สุดเพียงชิ้นเดียว

หากเจอวัตถุ:

คำนวณจุดศูนย์กลาง (Centroid)

วาดกรอบและจุดศูนย์กลางลงบนภาพ

ใช้ to_pos_robot() แปลงพิกัดเป็นของหุ่นยนต์

แสดงชื่อสีและพิกัดของหุ่นยนต์บนหน้าจอ

แสดงผลภาพที่วาดทับ (Annotated) และภาพ Mask

รอรับการกดปุ่มจากคีย์บอร์ด:

S: บันทึกการตั้งค่าปัจจุบัน

R: โหลดค่าที่บันทึกไว้ล่าสุด

P: พิมพ์พิกัดของวัตถุที่เจอในมุมมองกล้องออกมาทาง Terminal (สำหรับทำ Calibration)

Q/ESC: ออกจากโปรแกรม

main_combined.py

```python
import cv2
import numpy as np
import socket
import time
import json
import os
from threading import Thread

# ====== CONFIG ======
DEFAULT_HSV_RANGES = [
    {"name": "Yellow", "lower": (3,   137, 131), "upper": (47,  226, 220)},
    {"name": "Red",    "lower": (150, 147, 137), "upper": (226, 255, 196)},
    {"name": "Blue",   "lower": (87,   77,  63), "upper": (150, 250, 255)},
    {"name": "Green",  "lower": (36,   68, 114), "upper": (74,  219, 184)},
]
DRAW_COLORS = [(0,255,255), (0,0,255), (255,0,0), (0,255,0)]
MIN_AREA  = 200
BOX_TYPE  = 0
THICKNESS = 2

CONFIG_PATH = "hsv_config.json"

IP_ROBOT = "192.168.1.6"
PORT = 6601

# ====== Perspective Calibration ======
camera_points = np.float32([[585, 314], [243, 321], [563, 98], [239, 131]])
world_points = np.float32([[368.39, 93.74], [371.16, -29.02], [292.13, 83.77], [302.78, -29.97
matrix = cv2.getPerspectiveTransform(camera_points, world_points)
```

```python
def to_pos_robot(box1):
    camera_x, camera_y, r = box1[0], box1[1], box1[2]
    camera_coord = np.float32([[camera_x, camera_y]])
    robot_coord = cv2.perspectiveTransform(camera_coord.reshape(-1, 1, 2), matrix)
    robotx, roboty = robot_coord[0][0][0], robot_coord[0][0][1]
    return robotx, roboty, 90 - r

def compute_distortion_map(w, h, k):
    cx, cy = (w - 1) / 2.0, (h - 1) / 2.0
    max_rad = np.sqrt(cx**2 + cy**2)
    x = np.linspace(0, w - 1, w, dtype=np.float32)
    y = np.linspace(0, h - 1, h, dtype=np.float32)
    X, Y = np.meshgrid(x, y)
    Xc, Yc = X - cx, Y - cy
    xn, yn = Xc / max_rad, Yc / max_rad
    r2 = xn*xn + yn*yn
    scale = 1.0 + k * r2
    Xd = (xn * scale) * max_rad + cx
    Yd = (yn * scale) * max_rad + cy
    return Xd.astype(np.float32), Yd.astype(np.float32)

def ensure_odd(n): return max(1, n | 1)
def nothing(_): pass
```

```python
def load_config_or_default():
    if os.path.exists(CONFIG_PATH):
        with open(CONFIG_PATH, "r", encoding="utf-8") as f:
            data = json.load(f)
        return {
            "ranges": data.get("ranges", DEFAULT_HSV_RANGES),
            "distortion_kx100": int(data.get("distortion_kx100", 100)),
            "morph_op": int(data.get("morph_op", 0)),
            "kernel_size": int(data.get("kernel_size", 3)),
            "iterations": int(data.get("iterations", 1)),
        }
    else:
        return {
            "ranges": DEFAULT_HSV_RANGES,
            "distortion_kx100": 100,
            "morph_op": 0,
            "kernel_size": 3,
            "iterations": 1,
        }

def save_config(ranges, kx100, morph_op, ksize, iters):
    data = {
        "ranges": ranges,
        "distortion_kx100": int(kx100),
        "morph_op": int(morph_op),
        "kernel_size": int(ksize),
        "iterations": int(iters),
    }
    for r in data["ranges"]:
        r["lower"] = list(r["lower"])
        r["upper"] = list(r["upper"])
    with open(CONFIG_PATH, "w", encoding="utf-8") as f:
        json.dump(data, f, indent=2)
    print(f"Saved settings to {CONFIG_PATH}")
```

```python
def create_controls_windows(cfg):
    cv2.namedWindow("Controls", cv2.WINDOW_NORMAL)
    cv2.resizeWindow("Controls", 520, 400)
    cv2.createTrackbar("Distortion k x100", "Controls", cfg["distortion_kx100"], 200, nothing)
    cv2.createTrackbar("Morph Op (0-4)", "Controls", cfg["morph_op"], 4, nothing)
    cv2.createTrackbar("Kernel Size", "Controls", cfg["kernel_size"], 31, nothing)
    cv2.createTrackbar("Iterations", "Controls", cfg["iterations"], 10, nothing)
    cv2.createTrackbar("Start Mg400", "Controls", 0, 1, nothing)

    # per-color enable trackbars (allow multiple selection)
    ranges = cfg.get("ranges", DEFAULT_HSV_RANGES)
    for i, spec in enumerate(ranges):
        name = spec.get("name", f"Color{i}")
        track_name = f"Enable {name}"
        # default enable = 1 (enabled). Change as you like.
        cv2.createTrackbar(track_name, "Controls", 1, 1, nothing)

def read_controls(cfg):
    kx100 = cv2.getTrackbarPos("Distortion k x100", "Controls")
    morph = cv2.getTrackbarPos("Morph Op (0-4)", "Controls")
    ksize = ensure_odd(cv2.getTrackbarPos("Kernel Size", "Controls"))
    iters = cv2.getTrackbarPos("Iterations", "Controls")

    # read per-color enables
    enabled = []
    ranges = cfg.get("ranges", DEFAULT_HSV_RANGES)
    for i, spec in enumerate(ranges):
        name = spec.get("name", f"Color{i}")
        track_name = f"Enable {name}"
        try:
            val = cv2.getTrackbarPos(track_name, "Controls")
        except:
            val = 1
        enabled.append(bool(val))
    return kx100, morph, ksize, iters, enabled
```

ตั้งแต่เริ่มจนถึงฟังก์ชันนี้เหมือนกันกับ client.py

```python
class Mg400(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.daemon = True
        self.status = 'wait'
        self.pos_frame = None
        self.sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        self.sock.connect((IP_ROBOT, PORT))
        time.sleep(1)
        self.sock.send('hi'.encode())
        print('Connected to robot.')

    def run(self):
        while True:
            print(self.status)
            if self.status == 'wait':
                data = self.sock.recv(50)
                if data == b'start':
                    self.status = 'find'
                    time.sleep(1)
                elif data == b'pos?':
                    self.status = 'find_pos'
                    time.sleep(1)

            if self.status == 'find':
                if self.pos_frame:
                    self.sock.send('found'.encode())
                    print('found')
                    self.status = 'wait'
                else:
                    print("Not Found!!")
                time.sleep(1)

            if self.status == 'find_pos':
                print("Mg400 pos_frame:", self.pos_frame)
                if self.pos_frame:
                    x, y, r = to_pos_robot(self.pos_frame)
                    msg = f'{x:.2f},{y:.2f},{r:.2f}'
                    self.sock.send(msg.encode())
                    print(f'Sent: {msg}')
                    self.status = 'wait'
                else:
                    print('Not found')
                    self.sock.send('finish'.encode())
                time.sleep(1)

            time.sleep(0.1)
```

โมดูลสื่อสารกับหุ่นยนต์ MG400 (Robot Communication Thread)

ทำงานใน Class Mg400 ซึ่งเป็น Thread ที่จัดการการสื่อสารกับหุ่นยนต์ผ่าน TCP/IP Socket โดยเฉพาะ

__init__(): เมื่อ Thread เริ่มต้น จะทำการสร้าง Socket และเชื่อมต่อไปยัง IP ของหุ่นยนต์ (IP_ROBOT) และ Port ที่กำหนด

run(): เป็น Loop การทำงานหลักของ Thread นี้ ทำหน้าที่เป็น State Machine เพื่อรอรับคำสั่งจากหุ่นยนต์และตอบสนองตามสถานะต่างๆ:

status = 'wait': สถานะรอรับคำสั่งจากหุ่นยนต์

ถ้ารับข้อความ b'start' จากหุ่นยนต์ จะเปลี่ยนสถานะเป็น 'find'

ถ้ารับข้อความ b'pos?' จากหุ่นยนต์ จะเปลี่ยนสถานะเป็น 'find_pos'

status = 'find': หุ่นยนต์ต้องการรู้ว่า "เจอวัตถุหรือไม่?"

Thread จะตรวจสอบตัวแปร self.pos_frame ที่ Vision Thread อัปเดตให้

ถ้ามีข้อมูล (เจอวัตถุ) จะส่งข้อความ 'found' กลับไป

ถ้าไม่มีข้อมูล จะไม่ทำอะไร (หรือควรจะส่ง 'not found') แล้วกลับไปรอ

status = 'find_pos': หุ่นยนต์ต้องการ "พิกัดของวัตถุ"

Thread จะตรวจสอบ self.pos_frame อีกครั้ง

ถ้ามีข้อมูล จะนำพิกัด (pixel) มาเข้าฟังก์ชัน to_pos_robot() เพื่อแปลงเป็นพิกัดหุ่นยนต์

จากนั้นจัดรูปแบบข้อความเป็น f'{x:.2f},{y:.2f},{r:.2f}' แล้วส่งให้หุ่นยนต์

ถ้าไม่มีข้อมูล จะส่ง 'finish' กลับไป

หลังจากส่งข้อมูลเสร็จ จะกลับไปที่สถานะ 'wait'

```python
class VisionProcessing(Thread):
    def __init__(self, mg400, cfg):
        Thread.__init__(self)
        self.daemon = True
        self.mg400 = mg400
        self.cfg = cfg
        self.cap = cv2.VideoCapture(0, cv2.CAP_DSHOW) if hasattr(cv2, "CAP_DSHOW") else cv2.Vi
        self.kernel = np.ones((5, 5), np.uint8)
        self.last_w = self.last_h = -1
        self.last_k = None
        self.map_x = self.map_y = None
        self.start()
```

```python
def run(self):
    font = cv2.FONT_HERSHEY_SIMPLEX
    while True:
        ret, frame = self.cap.read()
        if not ret:
            continue

        h, w = frame.shape[:2]
        kx100, morph, ksize, iters, enabled = read_controls(self.cfg)
        k_val = (kx100 - 100) / 100.0

        # Recompute remap if needed
        if (w != self.last_w) or (h != self.last_h) or (k_val != self.last_k) or (self.map
            self.map_x, self.map_y = compute_distortion_map(w, h, k_val)
            self.last_w, self.last_h, self.last_k = w, h, k_val

        # Apply distortion ONCE
        distorted = cv2.remap(frame, self.map_x, self.map_y, interpolation=cv2.INTER_LINEA

        hsv = cv2.cvtColor(distorted, cv2.COLOR_BGR2HSV)
        annotated = distorted.copy()
        pos_frame_0 = []
        all_masks = []

        # iterate cfg ranges but only process enabled ones
        ranges = self.cfg.get("ranges", DEFAULT_HSV_RANGES)
        for i, spec in enumerate(ranges):
            is_enabled = enabled[i] if i < len(enabled) else True
            lower = np.array(spec["lower"], dtype=np.uint8)
            upper = np.array(spec["upper"], dtype=np.uint8)
            mask = cv2.inRange(hsv, lower, upper)

            # store mask for combined/single display
            all_masks.append(mask)

            if not is_enabled:
                # skip processing/drawing for disabled colors
                continue
```

```python
    # apply morphology and contour processing for enabled masks
    if ksize > 1 and iters > 0:
        kern = cv2.getStructuringElement(cv2.MORPH_RECT, (ksize, ksize))
        if morph == 1:    mask = cv2.erode(mask, kern, iterations=iters)
        elif morph == 2: mask = cv2.dilate(mask, kern, iterations=iters)
        elif morph == 3: mask = cv2.morphologyEx(mask, cv2.MORPH_OPEN, kern, iterat
        elif morph == 4: mask = cv2.morphologyEx(mask, cv2.MORPH_CLOSE, kern, itera

    contours, _ = cv2.findContours(mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE
    best_cnt = None
    best_score = 0.0
    for c in contours:
        area = cv2.contourArea(c)
        if area < MIN_AREA:
            continue
        x, y, ww, hh = cv2.boundingRect(c)
        box_area = max(1, ww * hh)
        fill_ratio = float(area) / float(box_area)
        score = area * fill_ratio
        if score > best_score:
            best_cnt = c
            best_score = score

    color = DRAW_COLORS[i % len(DRAW_COLORS)]
    label_name = spec.get("name", f"Color{i}")

    if best_cnt is not None:
        M = cv2.moments(best_cnt)
        if M["m00"] != 0:
            cx = int(M["m10"] / M["m00"])
            cy = int(M["m01"] / M["m00"])
        else:
            x, y, w0, h0 = cv2.boundingRect(best_cnt)
            cx, cy = x + w0 // 2, y + h0 // 2

        rect = cv2.minAreaRect(best_cnt)
        angle = rect[-1]
        if angle < -45:
            angle = 90 + angle
        angle = round(angle, 0)

        pos_frame_0.append([cx, cy, angle])
```

```
        if BOX_TYPE == 0:
            x, y, w0, h0 = cv2.boundingRect(best_cnt)
            cv2.rectangle(annotated, (x, y), (x + w0, y + h0), color, THICKNESS)
        else:
            box = cv2.boxPoints(rect)
            box = box.astype(np.int32)
            cv2.polylines(annotated, [box], True, color, THICKNESS)

        cv2.circle(annotated, (cx, cy), 4, (0,0,0), -1)
        cv2.circle(annotated, (cx, cy), 3, color, -1)
        robotx, roboty, robotr = to_pos_robot([cx, cy, angle])
        text = f"{label_name} Robot({robotx:.1f},{roboty:.1f}) angle:{robotr:.1f}"
        cv2.putText(annotated, text, (cx + 6, cy - 6), font, 0.6, (0,0,0), 3, cv2.
        cv2.putText(annotated, text, (cx + 6, cy - 6), font, 0.6, (255,255,255), 1

# Send the first found cube position to Mg400 thread
if pos_frame_0:
    #print("Detected cube:", pos_frame_0[0])
    self.mg400.pos_frame = pos_frame_0[0]
else:
    self.mg400.pos_frame = None

# show combined or per-mask view depending on '[' / ']' handling elsewhere
if all_masks:
    combined_mask = np.zeros_like(all_masks[0])
    for m in all_masks:
        combined_mask = cv2.bitwise_or(combined_mask, m)
    cv2.imshow("Combined Mask", combined_mask)

cv2.imshow("Annotated Output", annotated)
```

```
    key = cv2.waitKey(1) & 0xFF
    if key in (27, ord('q')):
        kx100, morph, ksize, iters, _ = read_controls(self.cfg)
        save_config(DEFAULT_HSV_RANGES, kx100, morph, ksize, iters)
        break
    elif key in (ord('s'), ord('S')):
        kx100, morph, ksize, iters, _ = read_controls(self.cfg)
        save_config(DEFAULT_HSV_RANGES, kx100, morph, ksize, iters)
    elif key in (ord('r'), ord('R')):
        print("Reset to defaults.")
        cv2.destroyAllWindows()
        self.cfg = load_config_or_default()
        create_controls_windows(self.cfg)

self.cap.release()
cv2.destroyAllWindows()
```

โมดูลประมวลผลภาพ (Vision Processing Thread)

ทำงานใน Class VisionProcessing ซึ่งเป็น Thread แยกออกมาเพื่อให้การประมวลผลภาพไม่ไปรบกวนการสื่อสารกับหุ่นยนต์

การทำงาน: มีโครงสร้างคล้ายกับ main() ใน client.py มาก คือ รับภาพ, แก้ไขความบิดเบี้ยว, แปลงเป็น HSV, สร้าง Mask, และค้นหาวัตถุที่ดีที่สุด

สิ่งที่แตกต่างและสำคัญ:

Enable/Disable Colors: ในหน้าต่าง "Controls" จะมี Trackbar เพิ่มขึ้นมาเพื่อ "เปิด/ปิด" การตรวจจับของ1 แต่ละสีได้ ทำให้สามารถเลือกได้ว่าจะให้หุ่นยนต์มองหาวัตถุสีอะไรบ้าง

การส่งข้อมูล: เมื่อตรวจพบวัตถุที่ต้องการ โปรแกรมจะเก็บข้อมูลตำแหน่งของวัตถุชิ้นแรกที่เจอ ([cx, cy, angle]) ไว้ในตัวแปร self.mg400.pos_frame ซึ่งเป็นตัวแปรที่ ใช้ร่วมกัน (Shared) กับ Thread ที่ทำหน้าที่สื่อสารกับหุ่นยนต์ นี่คือจุดเชื่อมต่อระหว่างสอง Thread

```python
def main():
    cfg = load_config_or_default()
    create_controls_windows(cfg)
    mg400_thread = Mg400()
    vision_thread = VisionProcessing(mg400_thread, cfg)
    started = False
    print("Controls: S = save, R = reset to defaults, Q/ESC = quit")
    try:
        while True:
            time.sleep(0.1)
            cv2.waitKey(1)  # <-- Add this line to keep the Controls window responsive
            start_val = cv2.getTrackbarPos("Start Mg400", "Controls")
            if start_val == 1 and not started:
                mg400_thread.start()
                print("Mg400 thread started!")
                started = True
    except KeyboardInterrupt:
        print("Exiting...")

if __name__ == "__main__":
    main()
```

โมดูลหลักในการควบคุมโปรแกรม (Main Execution)

main() ในไฟล์นี้ทำหน้าที่ตั้งค่าเริ่มต้นและจัดการ Thread ทั้งสอง

โหลดค่า Config ที่ได้จาก client.py

สร้างหน้าต่าง UI (คล้ายของเดิม แต่เพิ่ม Trackbar "Start Mg400" และ "Enable Colors")

สร้าง Instance ของ Mg400 (Communication Thread) และ VisionProcessing (Vision Thread)

เริ่มการทำงานของ Vision Thread ทันที (vision_thread.start())

รอ ให้ผู้ใช้เลื่อน Trackbar "Start Mg400" เป็น 1 จึงจะ เริ่มการทำงานของ Communication Thread (mg400_thread.start()) เพื่อให้ผู้ใช้เป็นคนตัดสินใจเริ่มการเชื่อมต่อกับหุ่นยนต์

Loop หลักของ main จะทำเพียงแค่ time.sleep และ cv2.waitKey เพื่อให้โปรแกรมทำงานต่อไปและหน้าต่าง UI ตอบสนองได้ โดยการทำงานหนักๆ จะเกิดขึ้นใน Thread ทั้งสองที่แยกกันทำงานอยู่เบื้องหลัง