1)
    a) Move the arm to the home position
    b) Detect the center of the Aruco blocks in the camera frame
    c) Calculate the center of each block in the table frame through the method described below
    d) Make sure the gripper is deactivated
    e) For each Aruco block:
        i) Use our IK code to calculate the joint angles required to move the arm above the block and move there
        ii) Move the arm down, activate the gripper, and move up
        iii) Use our IK code to calculate the joint angles required to move the arm to above the destination and move there
        iv) Move the arm down, deactivate the gripper, and move up
        v) Move the arm to the home position

2) First, get the perspective matrix from the camera frame to the table frame (done using the get_perspective_warping.py script). We can then left-multiply the marker centers in the camera frame by this perspective matrix, much like we would with a homogeneous transformation. However, because this is a non-homogeneous transformation, our 2D point is still not in the correct perspective. To rectify this, we divide the entire column vector, which represents our position in 2D space, by the scale factor (the third row in our column vector after multiplying by the perspective matrix), and now we have the position of the block relative to the table frame.

Videos:
Pick And Place: https://youtube.com/shorts/4KisNRZduw8?feature=share
Extra Credit: https://youtube.com/shorts/Xeba4YLFgU0?feature=share

1. Pseudo code for detecting and moving the block (no specific format to be followed)
2. Math for camera frame to table frame
3. Video of pick and place task on UR3e (as a link (GDrive/YouTube) in the report)
4. Extra Credit: Stacking (3.33 pts per block) - Max 10pts possible

```
import cv2
import numpy as np
import keyboard

# Mouse callback function
def mouse_callback(event, x, y, flags, param):
   global selected_points, frame

   if event == cv2.EVENT_LBUTTONDOWN and len(selected_points) < 4:
       selected_points.append((x, y))

   return frame

if __name__ == '__main__':
   selected_points = []
   frame = None
   # Turn on Laptop's webcam
   cap = cv2.VideoCapture("/dev/video0")

   ret, frame = cap.read()

   while ret:
       ret, frame = cap.read()

       if len(selected_points) > 0:
           points_in_image_frame = selected_points
           for ii in range(len(selected_points)):
               cv2.circle(frame, selected_points[ii], 3, (0, 0, 255), -1)

       ###################################### YOUR CODE STARTS HERE
######################################

       # Fill up the array with the points you have chosen. The sequence of points must be
similar to the sequence in which you will be clicking on the image!

       points_in_table_frame = np.array([(0,0), (0,350), (350,350), (350, 0)],
dtype=np.float32).reshape(4,2)

       ###################################### YOUR CODE ENDS HERE
######################################

       if len(selected_points) == 4:
           points_in_image_frame = np.array(points_in_image_frame,
dtype=np.float32).reshape(4,2)
```

```
            perspective_matrix = cv2.getPerspectiveTransform(points_in_image_frame,
points_in_table_frame)
            result = cv2.warpPerspective(frame, perspective_matrix, (550, 450))

            cv2.circle(result, (175, 175), 3, (255, 0, 0), -1)
            cv2.imshow('frame1', result) # Transformed Capture

        # Wrap the transformed image
        cv2.imshow('Camera Frame', frame) # Initial Capture
        frame = cv2.setMouseCallback("Camera Frame", mouse_callback)

        if cv2.waitKey(int(1 / 60 * 1000)) & 0xFF == ord('q'):
            np.save("perspective_matrix.npy", perspective_matrix)
            break

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

```
#!/usr/bin/env python
import cv2
import numpy as np
import cv2.aruco as aruco
import yaml

def detect_aruco(img, camera_matrix, distortion_coefficients):
    gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
    aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
    parameters = aruco.DetectorParameters_create()
    corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters = parameters)
    output = aruco.drawDetectedMarkers(img, corners, ids)  # detect the aruco markers and
display its aruco id.

    if ids is not None:
        for i in range(len(ids)):
            # Draw the detected aruco onto the frame
            rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[i], 0.0635, camera_matrix,
distortion_coefficients)
            output = aruco.drawAxis(output, camera_matrix, distortion_coefficients, rvec, tvec,
0.1)

    return output, ids, corners

def get_aruco_center(img, ids, corners):
    marker_centers = [(np.nan, np.nan)] * len(ids)
    for i in range(len(ids)):
        corner = corners[i][0, :, :]
        marker_centers[i] = np.mean(corner, axis=0)
```

```
        x = marker_centers[i][0]
        y = marker_centers[i][1]
        # Draw the detected marker center onto the frame with yellow color
        img = cv2.circle(img, (int(x), int(y)), 3, (0, 255, 255), -1)

    return img, ids, marker_centers

def image_frame_to_table_frame(img, ids, marker_centers, perspective_matrix):
    marker_centers_in_table_frame = [(np.nan, np.nan)] * len(ids)

    ############################ YOUR CODE STARTS HERE ###################################

    marker_centers = [np.append(marker_center, 1) for marker_center in marker_centers]

    for i in range(0, len(ids)):

        marker_centers_in_table_frame[i] = np.matmul(perspective_matrix,marker_centers[i])
        marker_centers_in_table_frame[i] /= marker_centers_in_table_frame[i][2]

        # Format the coordinates as strings for display

        value1 = str(marker_centers_in_table_frame[i][0])
        value2 = str(marker_centers_in_table_frame[i][1])

        # Display the information on the image
        img = cv2.putText(img, "Aruco ID: " + str(ids[i]), (400, 20 + 90*i),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2, cv2.LINE_AA, False)
        img = cv2.putText(img, "X (mm): " + value1, (400, 50 + 90*i), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 255, 0), 2, cv2.LINE_AA, False)
        img = cv2.putText(img, "Y (mm): " + value2, (400, 80 + 90*i), cv2.FONT_HERSHEY_SIMPLEX,
0.5, (0, 255, 0), 2, cv2.LINE_AA, False)

    ############################ YOUR CODE ENDS HERE #####################################

    return img, marker_centers_in_table_frame


def get_camera_calibration_params(path):
    # Get the camera calibration parameters from the .yaml file
    with open(path, 'r') as file:
        yaml_data = yaml.safe_load(file)
    cm = yaml_data['camera_matrix']['data']
    cm = np.array(cm)
    camera_matrix = cm.reshape(3,3)

    distortion_coefficients = yaml_data['distortion_coefficients']['data']
    distortion_coefficients = np.array(distortion_coefficients)
    distortion_coefficients = distortion_coefficients.reshape(1,5)
```

```
    return camera_matrix, distortion_coefficients


def GetStream(video_in, camera_matrix, distortion_coefficients, perspective_matrix):
    cap = cv2.VideoCapture(video_in)
    if not cap.isOpened():
        print("Error opening cam.")
        exit(0)

    rval, frame = cap.read()

    while rval:

        rval, frame = cap.read()
        key = cv2.waitKey(20)

        frame, ids, corners = detect_aruco(frame, camera_matrix, distortion_coefficients)
        frame, ids, marker_centers = get_aruco_center(frame, ids, corners)
        frame, marker_centers_in_table_frame = image_frame_to_table_frame(frame, ids,
marker_centers, perspective_matrix)

        cv2.imshow("Stream: cam 0: ", frame)
        if cv2.waitKey(10) & 0xFF == ord('q'):
            break

    cap.release()
    if save_video_flag:
        out.release()
    cv2.destroyWindow("Stream: cam 0: ")

if __name__ == '__main__':

    # Load the Camera to Base Transformation Matrix
    perspective_matrix =
np.load("/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/perspective_matri
x.npy")
    path_to_camera_matrix =
'/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/config/logitech_webcam_64
0x480.yaml'

    camera_matrix, distortion_coefficients =
get_camera_calibration_params(path_to_camera_matrix)
    cam_0 = "/dev/video0"
    GetStream(cam_0, camera_matrix, distortion_coefficients, perspective_matrix)
```

```
#!/usr/bin/env python3
import cv2
import numpy as np
import cv2.aruco as aruco
import yaml
import rclpy
from rclpy.node import Node
from sensor_msgs.msg import Image
from cv_bridge import CvBridge
from std_msgs.msg import String  # For publishing positions as text

class ArucoTracker(Node):
   def __init__(self, camera_matrix_path, perspective_matrix_path):
       super().__init__('aruco_tracker')

       # Load camera calibration parameters and perspective transformation matrix
       self.camera_matrix, self.distortion_coefficients =
self.load_camera_calibration(camera_matrix_path)
       self.perspective_matrix = np.load(perspective_matrix_path)

       # Initialize ROS2 subscribers, publishers, and CvBridge
       self.bridge = CvBridge()
       self.image_subscription = self.create_subscription(Image, '/camera1/image_raw',
self.image_callback, 10)
       self.image_publisher = self.create_publisher(Image, '/aruco_detection/image', 10)
       self.position_publisher = self.create_publisher(String, '/aruco_detection/positions',
10)

   @staticmethod
   def crop_frame(image):
       margin_up, margin_down, margin_left, margin_right = 230, 0, 60, 52
       h, w, _ = image.shape
       return image[margin_up:h-margin_down, margin_left:w-margin_right]

   def get_frame(self,video_in):
       cap = cv2.VideoCapture(video_in)
       if not cap.isOpened():
           print("Error opening cam.")
           exit(0)
       '''
       #cap.set(1, k_frame)
       rval, frame = cap.read()
       frame_filename = ""
       frame = CropFrame(frame)
       cv2.imwrite(frame_filename, frame)
       print("Obtaining current frame.")
       '''
       rval, frame = cap.read()
```

```
        print("Obtaining current frame.")

        return frame

    def load_camera_calibration(self, path):
        with open(path, 'r') as file:
            yaml_data = yaml.safe_load(file)
        cm = np.array(yaml_data['camera_matrix']['data']).reshape(3, 3)
        distortion_coefficients =
np.array(yaml_data['distortion_coefficients']['data']).reshape(1, 5)
        return cm, distortion_coefficients

    def detect_aruco(self, img):
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        aruco_dict = aruco.Dictionary_get(aruco.DICT_6X6_250)
        parameters = aruco.DetectorParameters_create()
        corners, ids, _ = aruco.detectMarkers(gray, aruco_dict, parameters=parameters)
        output = aruco.drawDetectedMarkers(img, corners, ids)

        if ids is not None:
            for i in range(len(ids)):
                rvec, tvec, _ = aruco.estimatePoseSingleMarkers(corners[i], 0.0635,
self.camera_matrix, self.distortion_coefficients)
                output = aruco.drawAxis(output, self.camera_matrix,
self.distortion_coefficients, rvec, tvec, 0.1)

        return output, ids, corners

    def get_aruco_center(self, img, ids, corners):
        marker_centers = [(np.nan, np.nan)] * len(ids)
        for i in range(len(ids)):
            corner = corners[i][0, :, :]
            marker_centers[i] = np.mean(corner, axis=0)
            x, y = marker_centers[i]
            img = cv2.circle(img, (int(x), int(y)), 3, (0, 255, 255), -1)
        return img, ids, marker_centers

    def image_frame_to_table_frame(self, img, ids, marker_centers, perspective_matrix):
        marker_centers_in_table_frame = [(np.nan, np.nan)] * len(ids)

        ################################## YOUR CODE STARTS HERE
################################################

        marker_centers = [np.append(marker_center, 1) for marker_center in marker_centers]

        for i in range(0, len(ids)):

            marker_centers_in_table_frame[i] = np.matmul(perspective_matrix,marker_centers[i])
```

```
            marker_centers_in_table_frame[i] /= marker_centers_in_table_frame[i][2]

            # Format the coordinates as strings for display

            value1 = str(marker_centers_in_table_frame[i][0])
            value2 = str(marker_centers_in_table_frame[i][1])

            # Display the information on the image
            img = cv2.putText(img, "Aruco ID: " + str(ids[i]), (400, 20 + 90*i),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 2, cv2.LINE_AA, False)
            img = cv2.putText(img, "X (mm): " + value1, (400, 50 + 90*i),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2, cv2.LINE_AA, False)
            img = cv2.putText(img, "Y (mm): " + value2, (400, 80 + 90*i),
cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 255, 0), 2, cv2.LINE_AA, False)
        ################################## YOUR CODE ENDS HERE
################################################

        return img, marker_centers_in_table_frame

    def image_callback(self, msg):
        # Convert ROS2 image message to OpenCV format
        # print(msg)
        frame = self.bridge.imgmsg_to_cv2(msg, desired_encoding='bgr8')

        # Process the frame
        frame, ids, corners = self.detect_aruco(frame)
        if ids is not None:
            frame, ids, marker_centers = self.get_aruco_center(frame, ids, corners)
            frame, _ = self.image_frame_to_table_frame(frame, ids, marker_centers)

        # Publish the processed image
        processed_image_msg = self.bridge.cv2_to_imgmsg(frame, encoding="bgr8")
        self.image_publisher.publish(processed_image_msg)

def main(args=None):
    rclpy.init(args=args)
    camera_matrix_path =
'/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/config/logitech_webcam_64
0x480.yaml'
    perspective_matrix_path =
'/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/perspective_matrix.npy'

    # Initialize and spin the Aruco tracker node
    tracker = ArucoTracker(camera_matrix_path, perspective_matrix_path)
    rclpy.spin(tracker)

    # Cleanup
    tracker.destroy_node()
    rclpy.shutdown()
```

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

```
import numpy as np
import sys
import math

class KinematicFunctions():


   def calculate_dh_transform(self, joint_positions):

       ######################################## Your Code Starts Here
########################################

       # TODO: DH parameters for UR3e. Modify these parameters according to the robot's
configuration # a , alpha, d, theta

       a = [0, -0.244, -0.213, 0, 0, -0.0535]
       alpha = [90, 0, 0, 90, -90, 0]
       d = [0.152, 0, 0, 0.131, 0.083, 0.151]
       theta_offset = [180, 0, 0, -90, 0, 0]

       matrices = []

       alpha_rad = [np.deg2rad(float(x)) for x in alpha]
       theta_offset_rad = [np.deg2rad(float(x)) for x in theta_offset]

       for i in range(0,6):
           theta = joint_positions[i] + theta_offset_rad[i]
           matrix = np.zeros((4,4))
           matrix[0][0] = math.cos(theta)
           matrix[0][1] = -math.sin(theta) * math.cos(alpha_rad[i])
           matrix[0][2] = math.sin(theta) * math.sin(alpha_rad[i])
           matrix[0][3] = a[i] * math.cos(theta)
           matrix[1][0] = math.sin(theta)
           matrix[1][1] = math.cos(theta) * math.cos(alpha_rad[i])
           matrix[1][2] = -math.cos(theta) * math.sin(alpha_rad[i])
           matrix[1][3] = a[i] * math.sin(theta)
           matrix[2][1] = math.sin(alpha_rad[i])
           matrix[2][2] = math.cos(alpha_rad[i])
           matrix[2][3] = d[i]
           matrix[3][3] = 1
           matrices.append(matrix)

       transform = matrices[0]
```

```
        for i in range(1,6):
            transform = np.matmul(transform,matrices[i])

        #our dh table is in base frame,
        #this translates from base to world frame
        transform[0][3] -= .15
        transform[1][3] += .15
        transform[2][3] += .01

        ############################ Your Code Ends Here
#####################################

        return transform


    def inverse_kinematics(self, xWgrip, yWgrip, zWgrip, yaw_WgripDegree):

        return_value = np.array([0, 0, 0, 0, 0, 0])

        ##################################### Your Code Starts Here
#####################################

        # TODO: Function that calculates an elbow up inverse kinematics solution for the UR3

        # Step 1: find gripper position relative to the base of UR3

        theta_5 = -(np.pi)/2

        xGrip = xWgrip + 0.15
        yGrip = yWgrip - 0.15
        zGrip = zWgrip - 0.01

        yawWgrip = np.deg2rad(yaw_WgripDegree)

        # Step 2: find x_cen, y_cen, z_cen

        x_cen = xGrip - (.0535 * np.cos(yawWgrip))
        y_cen = yGrip - (.0535 * np.sin(yawWgrip))
        z_cen = zGrip

        # Step 3: find theta_1

        r = ((x_cen ** 2) + (y_cen ** 2)) ** (1/2)
        theta_b = np.arcsin(0.131 / r)
        theta = np.arctan(y_cen / x_cen)
        theta_1 = theta - theta_b

        # Step 4: find theta_6
```

```
        theta_6 = (np.pi/ 2) - yawWgrip + theta_1


        # Step 5: find x3_end, y3_end, z3_end

        x3_end = x_cen + (np.cos(theta_1) * -0.083) + (np.sin(theta_1) * 0.131)
        y3_end = y_cen + (np.sin(theta_1) * -0.083) + (np.cos(theta_1) * -0.131)
        z3_end = z_cen + 0.052 + 0.092


        # Step 6: find theta_2, theta_3, theta_4


        a = (x3_end ** 2 + y3_end ** 2) ** (1/2)
        b = z3_end - 0.152
        c = (a ** 2 + b ** 2) ** (1/2)


        gamma = np.arctan(b/a)
        delta = np.arccos((.244 ** 2 + c ** 2 - .213 ** 2)/(2 * .244 * c))


        theta_2 = -(delta + gamma)
        theta_3 = np.pi - np.arccos((.244 ** 2 + .213 ** 2 - c ** 2 )/(2 * .244 * .213))
        theta_4 = gamma - np.arccos((c ** 2 + .213 ** 2 - .244 ** 2)/(2 * c * .213))

        ############################ Your Code Ends Here
#######################################

        # print theta values (in degree) calculated from inverse kinematics

        # print("Correct Joint Angles: ")
        # print(str(theta_1*180/np.pi) + " " + str(theta_2*180/np.pi) + " " + \
        #         str(theta_3*180/np.pi) + " " + str(theta_4*180/np.pi) + " " + \
        #         str(theta_5*180/np.pi) + " " + str(theta_6*180/np.pi))

        # obtain return_value from forward kinematics function
        return_value = [theta_1, theta_2, theta_3, theta_4, theta_5, theta_6]

        return return_value
```

```
#!/usr/bin/env python

import sys
import copy
import time
import yaml

# import math libraries
```

```
import math
import numpy as np

# import CV libraries
import cv2
import cv2.aruco as aruco


# import ROS libraries
import rclpy
from rclpy.node import Node

# import ROS message libraries

from std_msgs.msg import String
from sensor_msgs.msg import Image
from geometry_msgs.msg import Point
from cv_bridge import CvBridge, CvBridgeError

# import custom messages and functions
from ur3e_mrc.msg import PositionUR3e, CommandUR3e, GripperInput

from enme480_project.kinematic_functions import KinematicFunctions
from enme480_project.block_detection_aruco import ArucoTracker

KF = KinematicFunctions()
ik = KF.inverse_kinematics

#################### END OF IMPORT ########################

class UR3eController(Node):

   def __init__(self):
       super().__init__('ur3e_controller')
       self.home = [np.radians(85), np.radians(-45), np.radians(45), np.radians(-90),
np.radians(-90), np.radians(90)]
       self.current_position = self.home
       self.thetas = [0.0] * 6
       self.gripper_toggle_state = False
       self.current_position_set = False
       self.digital_in_0 = 0
       self.SPIN_RATE = 20  # Adjust as needed
       self.vel = 0.1
       self.accel = 0.1

       # Publishers and Subscribers
       self.pub_command = self.create_publisher(CommandUR3e, 'ur3/command', 10)
       self.sub_position = self.create_subscription(PositionUR3e, 'ur3/position',
self.position_callback, 10)
```

```
        self.sub_input = self.create_subscription(GripperInput, 'ur3/gripper_input',
self.input_callback, 10)

        # Timer (if needed)
        # self.timer = self.create_timer(1.0 / self.SPIN_RATE, self.timer_callback)

    def position_callback(self, msg):
        self.thetas = msg.position
        self.current_position = list(self.thetas)
        self.current_position_set = True

    def input_callback(self, msg):
        self.digital_in_0 = msg.dig_in & 1

    def move_arm(self, dest):

        '''
        CommandUR3e.msg:
                float64[] destination  ----> joint angles
                float64 v ----> velocity
                float64 a ----> acceleration
                bool io_0 ----> vacuum gripper input (True/False)
        '''

        ################## YOUR CODE STARTS HERE #################################

        command_msg = CommandUR3e()
        command_msg.destination = dest
        command_msg.v = 0.1
        command_msg.a = 0.1
        command_msg.io_0 = self.gripper_toggle_state
        self.pub_command.publish(command_msg)

        ################## YOUR CODE ENDS HERE ###################################

        # Wait until the robot reaches the goal
        while not self.at_goal(dest):
            rclpy.spin_once(self)
            time.sleep(0.05)  # Adjust sleep duration as needed
            pass

    def gripper_control(self, toggle_state):

        '''
        CommandUR3e.msg:
                float64[] destination  ----> joint angles
                float64 v ----> velocity
                float64 a ----> acceleration
                bool io_0 ----> vacuum gripper input (True/False)
```

```
        '''

        ################## YOUR CODE STARTS HERE ###################################

        command_msg = CommandUR3e()
        command_msg.destination = self.current_position
        command_msg.v = 0.1
        command_msg.a = 0.1
        command_msg.io_0 = toggle_state
        self.gripper_toggle_state = toggle_state
        self.pub_command.publish(command_msg)

        ################## YOUR CODE ENDS HERE #####################################

        # Wait until the robot reaches the goal
        while not self.at_goal(self.current_position):
            rclpy.spin_once(self)
            time.sleep(0.05)  # Adjust sleep duration as needed
            pass

    def at_goal(self, destination):
        tolerance = 0.0005
        return all(abs(self.thetas[i] - destination[i]) < tolerance for i in range(6))


class BlockMover:

    def __init__(self, ur3e_controller, aruco_tracker, dest_positions):
        self.ur3e_controller = ur3e_controller
        self.aruco_tracker = aruco_tracker
        self.dest_positions = dest_positions
        self.intermediate_height = 0.2

    def move_block(self, initial_position, final_position):

        '''
        initial_position & final_position are lists ---> [x, y, z, yaw]

        TODO: Define the sequence for moving one single block from one position to the other
        '''

        ################## YOUR CODE STARTS HERE ###################################

        self.ur3e_controller.gripper_control(False)
        self.ur3e_controller.move_arm(ik(initial_position[0], initial_position[1],
initial_position[2] + 0.06, initial_position[3]))
        self.ur3e_controller.move_arm(ik(initial_position[0], initial_position[1],
initial_position[2], initial_position[3]))
        time.sleep(1)
```

```
        self.ur3e_controller.gripper_control(True)
        time.sleep(1)
        self.ur3e_controller.move_arm(ik(initial_position[0], initial_position[1],
initial_position[2] + 0.06, initial_position[3]))
        self.ur3e_controller.move_arm(ik(final_position[0], final_position[1],
final_position[2]+ 0.06, final_position[3]))
        self.ur3e_controller.move_arm(ik(final_position[0], final_position[1],
final_position[2], final_position[3]))
        self.ur3e_controller.gripper_control(False)
        self.ur3e_controller.move_arm(ik(final_position[0], final_position[1],
final_position[2]+ 0.06, final_position[3]))

        ################## YOUR CODE ENDS HERE ###################################

    def process_blocks(self, video_device, ids, destination):

        '''
        This function is used for processing the Aruco Markers, find their centers and convert
them to table frame. Once that's done, the function decides the sequence of block picking and
end destination for each block.

        TODO: Detect the aruco tags, find their centers, and convert the center position to
image frame.
              Then, strategize on how you want to move the blocks (eg. different groups, single
stack, grouped stacks). You can include all of them in this function if you want to try
multiple methods.

              You can use the move_block function defined above. Before moving the blocks, send
a command to move the robot to home position defined in the UR3eController class

            Below are the corresponding Aruco IDs for each color in RAL
                    block_color = Yellow    - id: 100
                    block_color = Red       - id: 150
                    block_color = Blue      - id: 200

        '''

        frame = self.aruco_tracker.get_frame(video_device)

        ################## YOUR CODE STARTS HERE #################################

        self.ur3e_controller.move_arm(self.ur3e_controller.home)

        frame, ids, corners = self.aruco_tracker.detect_aruco(frame)
        if ids is not None:
            frame, ids, marker_centers = self.aruco_tracker.get_aruco_center(frame, ids,
corners)
```

```
            frame, marker_centers_in_table_frame =
self.aruco_tracker.image_frame_to_table_frame(frame, ids, marker_centers,
self.aruco_tracker.perspective_matrix)

            for i in range(len(marker_centers_in_table_frame)):
                self.move_block([marker_centers_in_table_frame[i][0] / 1000, \
                                 marker_centers_in_table_frame[i][1] / 1000,
                                 0.055, 0], destination[i])
                self.ur3e_controller.move_arm(self.ur3e_controller.home)

        self.ur3e_controller.move_arm(self.ur3e_controller.home)

        ################## YOUR CODE ENDS HERE ###################################

def main():

    rclpy.init()

    try:

        # # Create an instance of UR3eController
        ur3e_controller = UR3eController()

        aruco_tracker = ArucoTracker(

camera_matrix_path='/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/config
/logitech_webcam_640x480.yaml',

perspective_matrix_path='/home/enme480_docker/ENME480_ws/src/enme480_project/enme480_project/p
erspective_matrix.npy'
        )

        # Define destination positions for each block. These are recommended sorting positions
for different colors. The given z is the lowest level in the stack
        dest_pose = [
            (0.1, -0.1, 0.052, 0), # Yellow Drop Off
            (0.102, -0.102, 0.093, -45), # Red Drop Off
            (0.102, -0.102, 0.134, 0), # Blue Drop Off
        ]

        # Initialize BlockMover and start moving blocks
        block_mover = BlockMover(ur3e_controller, aruco_tracker, dest_pose)
        block_ids = [100, 150, 200]  # Block IDs to move
        block_mover.process_blocks("/dev/video0", block_ids, dest_pose)


    except KeyboardInterrupt:

        ur3e_controller.get_logger().info("Test interrupted by user.")
```

```
    finally:

        # Clean up and shut down
        ur3e_controller.destroy_node()
        rclpy.shutdown()

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

Pseudocode for `process_blocks`

1. **Initialize Environment**
    - Move the robot to its **home position**.
2. **Capture Frame from Camera**
    - Retrieve a video frame using the Aruco Tracker (`aruco_tracker`) from the specified video device.
3. **Detect Aruco Markers**
    - Use `detect_aruco` to detect Aruco markers in the frame.
        - If no markers are detected, exit or wait for the next frame.
4. **Extract Marker Details**
    - For detected markers, calculate their centers (`get_aruco_center`).
    - Convert these center coordinates from the **image frame** to the **table frame** using the perspective matrix (`image_frame_to_table_frame`).
5. **Move Blocks**
    - Iterate over the list of detected markers:
        - For each marker:
            - Convert the **table frame position** to **robot coordinates** (x, y, z).
            - Use the `move_block` method to move the block from its current position to its assigned destination position.
6. **Return to Home Position**
    - After moving each block, return the robot to its **home position**.
7. **Post-processing**
    - Once all blocks are moved, ensure the robot is in the home position for safety and shutdown.