Lab 4 Write Up and Block Diagram

Gary Banks Dan Blanchette

October 26, 2023

1 Introduction

This document contains a technical explanation of Lab 4's code implementation. Our solution utilizes MQTT to transmit boolean flag data and the random cartesian joint offsets to pass a die between each robot. The criteria have the robots receiving random offset data from our partner's computer and adjusting each hand-off position accordingly. As part of our implementation, the robots will hand off the dice three times, with the final handoff to Bill depositing the payload in robot B's designated zone.

2 DJ(Robot A) Routine and Feedback Explanation

We chose DJ to be Robot A for our project. DJ is the first to pick up the dice payload and wait to hand it off to Bill (Robot B). This is achieved using a boolean flag dictionary, then formatted into a JSON string and passed to the MQTT broker via publication to the topic `"flag_data."` Bill will look for the `DJ_waiting and DJ_has_dice` flags. Once my laptop receives the flag `Bill_has\_dice` from my subscriber function, these flags are reset. The offset coordinates are published before Bill moves so that the cartesian calculations can be evaluated. Once the handoff is successful, DJ retracts on the Y-axis -500mm. My program then polls Bill to see if `Bill_waiting and Bill_has_dice` are set to true. Before those flags are set, my program subscribes and receives Bill's cartesian offsets for x, y, and z. The program adds the changes to the hand-off default reference point we initially set by manually teaching the robots to a hand-off pose. The remaining part of the program runs the routine so that DJ has exchanged the dice 3 times with Bill and deposited the payload twice.

2.1 DJ’s Code (Dan)

```python

# Advanced Robotics 2(CS 553)
# Author: Dan Blanchette
# Date: 10/23/23
# Lab 4: Dice Passing Robots
# Ver: 2.0

# Description: This program will use MQTT to communicate cartesian offsets and robot "states"
# to my partner Gary Bank’s program. We use Boolean flags to indicate feedback to the other robot
# for grabbing the dice and when each robot arrives at a random walk position. The dice are then passed
# to his robot, which will do its random walk handoff and send me the new coordinate offset via the MQTT
# broker. # This program utilizes the University of Idaho Fanuc Python API driver, Paho MQTT, and Mosquitto
# acting as a local broker on Gary’s laptop.


import os
import random
from robot_controller import robot
import paho.mqtt.client as mqtt
import json
import random
import time

# MQTT server details
BROKER_IP = "129.101.98.194"
BROKER_PORT = 1883
# DJ IP ADDRESS
drive_path = ’129.101.98.215’ # DJ

# Dictionary for Robot Hand off Start Location Pose
def_cart_data = {
    "x": 364.646,
    "y": 690.701,
    "z": 376.777,
    "w":-90.995,
    "p":-31.562,
   "r":-1.412
}

# Dictionary that will be used to update offset information for cartesian random walk
cart_data = {
    "x": 0.0,
    "y": 0.0,
    "z": 0.0
}

flag_data = {
    "dj_waiting": False,
    "dj_has_die": False,
    "bill_waiting": False,
    "bill_has_die": False
}

# robot API class instance
crx10_dj = robot(drive_path)

def on_publish(client, userdata, mid):
    print(f’Message Published: {userdata}’)


# Connect to broker verification
def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    client.subscribe("flag_data")
    client.subscribe("cart_data")

# Disconnect from Broker
def on_disconnect(client, userdata, rc, properties=None):
    print(f"Disconnected with result code {rc}")

# Topics to Subscribe to
def on_message(client, userdata, msg):
    if msg.topic == "cart_data":
        received_data = json.loads(msg.payload.decode())
        cart_data[’x’] = received_data.get(’x’, cart_data[’x’])  # if first value DNE, grab second value
        cart_data[’y’] = received_data.get(’y’, cart_data[’y’])
        cart_data[’z’] = received_data.get(’z’, cart_data[’z’])

    if msg.topic == "flag_data":
        received_data = json.loads(msg.payload.decode())
        flag_data[’dj_waiting’] = received_data.get(’dj_waiting’, flag_data[’dj_waiting’])
        flag_data[’dj_has_die’] = received_data.get(’dj_has_die’, flag_data[’dj_has_die’])
        flag_data[’bill_waiting’] = received_data.get(’bill_waiting’, flag_data[’bill_waiting’])
        flag_data[’bill_has_die’] = received_data.get(’bill_has_die’, flag_data[’bill_has_die’])

# MQTT Client Setup
client = mqtt.Client()
client.on_publish = on_publish
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message
client.connect(BROKER_IP, BROKER_PORT)
client.loop_start()



def main():

# main program
   # Local vars that hold the home and payload approach joint poses
```

```
    home = [3.6055996417999268,-1.542962312698364З,3.3683128356933594,
    -0.713886559009552,-4.529087066650391,-2.439002752304077]
    def_loc_grab = [17.481, 25.178, -51.212, 0.697,-38.636, 13.036]

    # Start Robot Routine
    # Open the Gripper
    crx10_dj.shunk_gripper(’open’)
    # Go to the home position
    crx10_dj.write_joint_pose(home)
    crx10_dj.start_robot()

    # Move to pick up dice position
    crx10_dj.write_joint_pose(def_loc_grab)
    crx10_dj.start_robot()

    # DEBUG TESTING VAR CHECK
    # print(f’DJ is moving:{move_flag}’)
    # print(f’{moving}’)

    # Close the gripper (non-blocking)
    crx10_dj.shunk_gripper(’close’)
    # Update the has die flag and publish to topic for Gary to subscribe to
    flag_data["dj_has_die"] = True
    message = json.dumps(flag_data)
    # publish grab flag as true and send to Gary’s lappy
    client.publish("flag_data", message, qos=1)
    print(f’I just sent Gary This Value:{message}’)

# Applying random offset to dictionary
    cart_data["x"] = random.uniform(-50.0, 50.0)
    print(f’x_off: {cart_data["x"]}’)
    cart_data["y"] = random.uniform(-50.0, 50.0)
    print(f’y_off: {cart_data["y"]}’)
    cart_data["z"] = random.uniform(-90.0, 90.0)
    print(f’z_off: {cart_data["z"]}’)
    # Sends offset data to Gary
    message2 = json.dumps(cart_data)
    client.publish("cart_data", message2, qos=2)

    # Add the random offset to the default point of reference for the hand off
    crx10_dj.write_cartesian_position(def_cart_data["x"] + cart_data["x"], def_cart_data["y"] + cart_data["y"], def_cart_data["z"] + cart_data["z"],
                                      def_cart_data["w"], def_cart_data["p"], def_cart_data["r"])
    # move to that position
    crx10_dj.start_robot()

    # Send Gary’s computer the waiting to hand off flag set as True
    flag_data["dj_waiting"] = True
    message1 = json.dumps(flag_data)
    print("DJ is waiting to hand off")
    client.publish("flag_data", message1, qos=1)

    # Poll Bill to see if robot has dice
    while(1):
    # Bill has the dice
        if(flag_data["bill_has_die"] == True):
            # DJ Opens gripper
            crx10_dj.shunk_gripper("open")
            # Reset DJ has dice flag to False
            flag_data['dj_has_die'] = False
            # Send Flag update to Gary's PC
            message2 = json.dumps(flag_data)
            client.publish("flag_data", message2, qos=1)

            #if DJ has dice is false, ok to -500mm y-axis retract
            if(flag_data["dj_has_die"]== False):
                print("Retracting.........")
                crx10_dj.write_cartesian_position(364.646, 190.701, 376.777, -90.995, -31.562, -1.412)
                crx10_dj.start_robot()
                time.sleep(0.2)
                break
        else:
            # debug statements
            print("waiting for Bill to grab")
            print(f'Bill Status:{flag_data["bill_has_die"]}')
            time.sleep(3)

    # Poll to see if Bill has die and Bill is waiting

    while(1):

        if(flag_data["bill_has_die"] == True and flag_data["bill_waiting"] == True):
            # move y+ 10mm to grab dice
            crx10_dj.write_cartesian_position(def_cart_data["x"] + cart_data["x"], def_cart_data["y"] + cart_data["y"] + 10, def_cart_data["z"] + cart_data["z"])
            crx10_dj.start_robot()
            # close the gripper
            crx10_dj.shunk_gripper('close')
            # set flags
            flag_data["dj_has_die"] = True
            flag_data["dj_waiting"] = False
            # update gripper flag is closed and DJ has the dice
            # and isn't waiting any more.
            message3 = json.dumps(flag_data)
            client.publish("flag_data", message3, qos=1)
            break


        else:
            print("Waiting for Bill to let go")
            print(f'Bill Dice Status:{flag_data["bill_has_die"]}')
            print(f'Bill Waiting Status:{flag_data["bill_waiting"]}')
            time.sleep(3)

    # wait for false flag from bill's gripper
    while(1):
        print(f’Bill has die == {flag_data["bill_has_die"]}’)
        if (flag_data["bill_has_die"] == False):
            # ok to move to default pick up/drop off position
            crx10_dj.write_joint_pose(def_loc_grab)
            crx10_dj.start_robot()

            # DICE CAN BE STICKY WHEN PLACING
            # Wait 1.5 seconds after opening then go home
            crx10_dj.shunk_gripper(’open’)
            time.sleep(1.5)
            crx10_dj.write_joint_pose(home)
            crx10_dj.start_robot()
            break


# REPETION 2

    # Go to the home position
    crx10_dj.write_joint_pose(home)
    crx10_dj.start_robot()

    # Get the dice from the default
    crx10_dj.write_joint_pose(def_loc_grab)
    crx10_dj.start_robot()

    # print statements for debugging flag data
    # print(f’DJ is moving:{move_flag}’)

    # close gripper
    crx10_dj.shunk_gripper(’close’)
    # set DJ has die flag to True
    flag_data["dj_has_die"] = True
    # Convert to JSON string and Publish Flag Data to Broker
    message = json.dumps(flag_data)
    # publish grab flag as true and send to Gary’s lappy
    client.publish("flag_data", message, qos=1)
    # Runtime Debugging Print Statement
    print(f’I just sent Gary This Value:{message}’)

# Applying random offset to dictionary
    cart_data["x"] = random.uniform(-50.0, 50.0)
    print(f’x_off: {cart_data["x"]}’)
    cart_data["y"] = random.uniform(-50.0, 50.0)
    print(f’y_off: {cart_data["y"]}’)
    cart_data["z"] = random.uniform(-90.0, 90.0)
    print(f’z_off: {cart_data["z"]}’)
    # Sends offset data to Gary
    message2 = json.dumps(cart_data)
    client.publish("cart_data", message2, qos=2)

    # Add received offset from Bill and add to default reference values
    crx10_dj.write_cartesian_position(def_cart_data["x"] + cart_data["x"], def_cart_data["y"] + cart_data["y"], def_cart_data["z"] + cart_data["z"],
                                     def_cart_data["w"],
    def_cart_data["p"], def_cart_data["r"])
    crx10_dj.start_robot()


    flag_data["dj_waiting"] = True
    message1 = json.dumps(flag_data)
    print("DJ is waiting to hand off")
    client.publish("flag_data", message1, qos=1)

    # Poll Bill to see if robot has dice
    while(1):
    # Bill has the dice
        if(flag_data["bill_has_die"] == True):
            # DJ Opens gripper
            crx10_dj.shunk_gripper("open")
            # Reset DJ has dice flag to False
            flag_data[’dj_has_die’] = False
            # Send Flag update to Gary’s PC
            message2 = json.dumps(flag_data)
            client.publish("flag_data", message2, qos=1)

            #if DJ has dice is false, ok to -500mm y-axis retract
            if(flag_data["dj_has_die"]== False):
                # From current position, retract 500mm on y-axis
                crx10_dj.write_cartesian_position(def_cart_data["x"
    ], def_cart_data["y"] - 500, def_cart_data["z"],
                                                  def_cart_data["w"
    ], def_cart_data["p"], def_cart_data["r"])
                crx10_dj.start_robot()
                time.sleep(0.5)
                print("Done Going Home Now.........")
                crx10_dj.write_joint_pose(home)
                crx10_dj.start_robot()
                break
        else:
            # debug statements
            print("waiting for Bill to take dice")
            print(f’Bill Status:{flag_data["bill_has_die"]}’)
            time.sleep(3)


if __name__=="__main__":
    main()

client.loop_stop()
```

3 Bills Routine and Feedback Explanation (Gary)

Bill’s program is based on subscribing to and publishing seven MQTT data points that Dan and I used to trigger robot movies and track states. These consisted of flags per robot for “is_waiting” and “has_die” and an XYZ offset used to communicate a cartesian offset from a predetermined reference handoff position. Bill moves to a “handoff approach” pose, where he sets his “is_waiting” flag and polls DJ's two flags to determine when DJ is ready to handoff the die. When those two flags are set, DJ clears his "is_waiting", applies the cartesian offset to his handoff position, moves in, grasps the die, and updates flags to communicate he has the die and is waiting. He then polls DJ to know when DJ has released, and the handoff has been completed. In this manner, Bill continues to move, poll, and update flags until the cycle has been completed.

4 Bill's Code (Gary)

```
#   Gary Banks
#   Robotics I
#   Lab 4
#
#
#
#   Command to start local mqtt broker
#      /usr/local/sbin/mosquitto -c /Users/gary/Documents/code/robot/lab4/mosquitto.conf
#
#


import random
from robot_controller import robot
import paho.mqtt.client as mqtt
import time
import json
import FANUCethernetipDriver

# MQTT server details
BROKER_IP = "129.101.98.195"
BROKER_PORT = 1883

# This dictionary contains the cartesian offset from our handoff reference point.
# It will be updated with random values, and published to a "cart_data" MQTT topic
cart_data = {
    "x": 0.0,
    "y": 0.0,
    "z": 0.0,
}

# This dictionary will contain the cartesian offset from our handoff reference point.
# It will be updated with random values, and published to a "cart_data" MQTT topic
flag_data = {
    "dj_waiting": False,
    "dj_has_die": False,
    "bill_waiting": False,
    "bill_has_die": False
}

```

```python

def on_publish(client, userdata, mid):
    print("Message Published...")


def on_connect(client, userdata, flags, rc):
    print(f"Connected with result code {rc}")
    client.subscribe("flag_data")
    client.subscribe("cart_data")


def on_disconnect(client, userdata, rc, properties=None):
    print(f"Disconnected with result code {rc}")


def on_message(client, userdata, msg):
    if msg.topic == "cart_data":
        print("Message received")

        # DEBUGGING
        # print("Received payload:", msg.payload.decode())

        # Decode the JSON
        received_data = json.loads(msg.payload.decode())

        # Update cart Values
        cart_data.update(received_data)

        # DEBUGGING print types and values
        print("FROM ON MESSAGE Cartesian values after Random x:", cart_data["x"], "(", type(cart_data["x"]), ")",
              " y:", cart_data["y"], "(", type(cart_data["y"]), ")",
              " z:", cart_data["z"], "(", type(cart_data["z"]), ")")

    if msg.topic == "flag_data":
        received_data = json.loads(msg.payload.decode())
        flag_data['dj_waiting'] = received_data.get('dj_waiting', flag_data['dj_waiting'])
        flag_data['dj_has_die'] = received_data.get('dj_has_die', flag_data['dj_has_die'])
        flag_data['bill_waiting'] = received_data.get('bill_waiting', flag_data['bill_waiting'])
        flag_data['bill_has_die'] = received_data.get('bill_has_die', flag_data['bill_has_die'])

# MQTT Setup
client = mqtt.Client()
client.on_publish = on_publish
client.on_connect = on_connect
client.on_disconnect = on_disconnect
client.on_message = on_message
client.connect(BROKER_IP, BROKER_PORT)

drive_path = '129.101.98.214'  # CRX10 BILL

```

```python
# Pose and cartesian position information
pose1 = [0.0, -2.409282387816347e-06, 0.009522347711026669, -0.02475867606994667, -0.018449142575263977,
          0.0247572660446167]  # Home position
pose2 = [-9.638092994689941, -9.305192947387695, -11.088014602661133, -1.9604847431182861, -78.9268798828125,
          -80.95472717285156]  # Approach Handoff
pose3 = [-38.522300720214844, 6.566395282745361, -11.665398597717285, -1.7110475301742554, -79.27543640136719,
          -52.11516189575195]  # Handoff
pose4 = [22.032983779907227, 30.85733985900879, -84.4652328491211, -21.550495147705078, 88.06071472167969,
          -177.08120727539062] # Die Drop-off
pose5 = [21.896, 17.596, -68.232, -22.433, 72.983, -170.967] # ABOVE Die Drop-off

handoff = [400.062, -491.382, 444.861, -179.717, 1.903, -90.965]  # cartesian

def main():
    """! Main program entry"""

    # DEBUGGING
    # print("Cartesian values x:", cart_data["x"], " y:", cart_data["y"], " z:", cart_data["z"])

    # MQTT stuff
    client.loop_start()

    # Create new robot object
    crx10 = robot(drive_path)

    # Set robot speed
    crx10.set_speed(200)

    # Open Gripper
    crx10.onRobot_gripper_close(90, 20)

    # #-----------------------------------------POSE ADJUST
    #
    # # Move arm lift die off belt
    # crx10.set_pose(pose4)
    # crx10.start_robot()
    #
    # exit()
    # #-----------------------------------------POSE ADJUST

    # Move arm HOME
    crx10.set_pose(pose1)
    crx10.start_robot()

    # Move arm to HANDOFF APPROACH
    crx10.set_pose(pose2)
    crx10.start_robot()

    # Loop at HANDOFF APPROACH, wait for DJ to be waiting and have die.
    while (1):
        if flag_data["dj_waiting"] == True and flag_data["dj_has_die"] == True:
            # PRINT DEBUGGING WITH TYPE INFO
            print("FROM ON MESSAGE Cartesian values after Random x: ", cart_data["x"], "(", type(cart_data["x"]), ")",
                  " y:", cart_data["y"], "(", type(cart_data["y"]), ")",
                  " z:", cart_data["z"], "(", type(cart_data["z"]), ")")

            # copy handoff cartesian, apply new cart_data from DJ to it
            temp_handoff = handoff.copy()  # What the heck

            # check cart_datas
            temp_handoff[0] += cart_data["x"]
            temp_handoff[1] += cart_data["y"]
            temp_handoff[2] += cart_data["z"]

            # Move robot to handoff+random, y - 80
            crx10.send_coords(temp_handoff[0], temp_handoff[1]+80, temp_handoff[2], temp_handoff[3], temp_handoff[4],
                              temp_handoff[5])
            crx10.start_robot()
            # Move to complete handoff + random
            crx10.send_coords(temp_handoff[0], temp_handoff[1], temp_handoff[2], temp_handoff[3], temp_handoff[4],
                              temp_handoff[5])
            crx10.start_robot()
            break
        else:
            print("Waiting for DJ")
            print("DJ flag data waiting:", flag_data["dj_waiting"], " die:", flag_data["dj_has_die"])
            time.sleep(1)

    # grasp die
    crx10.onRobot_gripper_close(77, 15)
    # Wait for grasp NEEDS TO BE OPTIMIZED
    time.sleep(4)
    # Tell DJ I have the die and I’m waiting for him
    flag_data[’bill_has_die’] = True
    flag_data[’bill_waiting’] = True
    message = json.dumps(flag_data)
    client.publish("flag_data", message, qos=1)

    # Goto approach handoff, wait for DJ to release die
    while 1:
        if flag_data["dj_has_die"] == False:

            # Tell DJ I have the die and I’m waiting for him
            flag_data[’bill_has_die’] = True
            flag_data[’bill_waiting’] = False
            message = json.dumps(flag_data)
            client.publish("flag_data", message, qos=1)

            time.sleep(5)

            # go to ABOVE pickup die
            crx10.set_pose(pose5)
            crx10.start_robot()
            # Go drop die off
            crx10.set_pose(pose4)
            crx10.start_robot()

            break
        else:
            print("Waiting for DJ to release")
            print("DJ flag data waiting:", flag_data["dj_waiting"], " die:", flag_data["dj_has_die"])
            time.sleep(.5)

    # Open Gripper, drop die in square
    crx10.onRobot_gripper_close(90, 20)

    # go to ABOVE pickup die
    crx10.set_pose(pose5)
    crx10.start_robot()

    # go home
    crx10.set_pose(pose1)
    crx10.start_robot()

    # Generate a random value between -50 and 50 for x and y, 100 for z
    cart_data[’x’] = round(random.uniform(-50.0, 50.0), 3)
    cart_data[’y’] = round(random.uniform(-50.0, 50.0), 3)
    cart_data[’z’] = round(random.uniform(-90.0, 100.0), 3)

    # Publish Random offset
    cart_message = json.dumps(cart_data)
    client.publish("cart_data", cart_message, qos=1)

    # Reset temp_handoff
    temp_handoff = handoff.copy()  # What the heck
    # Apply random
    temp_handoff[0] += cart_data[’x’]
    temp_handoff[1] += cart_data[’y’]
    temp_handoff[2] += cart_data[’z’]

    # go to ABOVE pickup die
    crx10.set_pose(pose5)
    crx10.start_robot()

    # go to pickup die
    crx10.set_pose(pose4)
    crx10.start_robot()

    # CloseGripper
    crx10.onRobot_gripper_close(77, 15)

    # go approach handoff
    crx10.set_pose(pose2)
```

```
crx10.start_robot()

# Move robot to handoff+random, y - 80
crx10.send_coords(temp_handoff[0], temp_handoff[1] + 80, temp_handoff[2], temp_handoff[3], temp_handoff[4],
                  temp_handoff[5])
crx10.start_robot()
# Move to complete handoff + random
crx10.send_coords(temp_handoff[0], temp_handoff[1], temp_handoff[2], temp_handoff[3], temp_handoff[4],
                  temp_handoff[5])
crx10.start_robot()

flag_data[’bill_has_die’] = True
flag_data[’bill_waiting’] = True
message = json.dumps(flag_data)
client.publish("flag_data", message, qos=1)

# Wait for DJ to grasp, release
while (1):
    if (flag_data["dj_has_die"] == True):
        # OpenGripper
        crx10.onRobot_gripper_close(90, 20)
        time.sleep(3)
        # Tell DJ I have the die and I’m waiting for him
        flag_data[’bill_has_die’] = False
        flag_data[’bill_waiting’] = False
        message = json.dumps(flag_data)
        client.publish("flag_data", message, qos=1)

        break
    else:
        print("Waiting for DJ to grasp")
        print("DJ flag data waiting:", flag_data["dj_waiting"], " die:", flag_data["dj_has_die"])
        time.sleep(.5)

# Move robot to handoff+random, y - 80
crx10.send_coords(temp_handoff[0], temp_handoff[1] + 80, temp_handoff[2], temp_handoff[3], temp_handoff[4],
                  temp_handoff[5])
crx10.start_robot()

# Move arm to handoff approach
crx10.set_pose(pose2)
crx10.start_robot()

# -------------------------------------------END OF LOOP !!!!!!

# Goto approach handoff, wait for DJ
while (1):
    if flag_data["dj_waiting"] == True and flag_data["dj_has_die"] == True:
        # PRINT DEBUGGING WITH TYPE INFO
        print("FROM ON MESSAGE Cartesian values after Random x: ", cart_data["x"], "(", type(cart_data["x"]), ")",
              " y:", cart_data["y"], "(", type(cart_data["y"]), ")",
              " z:", cart_data["z"], "(", type(cart_data["z"]), ")")

        # copy handoff cartesian, apply new cart_data from DJ to it
        temp_handoff = handoff.copy()  # What the heck

        # check cart_datas
        temp_handoff[0] += cart_data["x"]
        temp_handoff[1] += cart_data["y"]
        temp_handoff[2] += cart_data["z"]

        # Move robot to handoff+random, y - 80
        crx10.send_coords(temp_handoff[0], temp_handoff[1] + 80, temp_handoff[2], temp_handoff[3], temp_handoff[4],
                          temp_handoff[5])
        crx10.start_robot()
        # Move to complete handoff + random
        crx10.send_coords(temp_handoff[0], temp_handoff[1], temp_handoff[2], temp_handoff[3], temp_handoff[4],
                          temp_handoff[5])
        crx10.start_robot()
        break
    else:
        print("Waiting for DJ")
        print("DJ flag data waiting:", flag_data["dj_waiting"], " die:", flag_data["dj_has_die"])
        time.sleep(1)

# #grasp die
crx10.onRobot_gripper_close(77, 15)
# Wait for grasp
time.sleep(3)
# Tell DJ I have the die and I'm waiting for him
flag_data['bill_has_die'] = True
flag_data['bill_waiting'] = True
message = json.dumps(flag_data)
client.publish("flag_data", message, qos=1)

# Goto approach handoff, wait for DJ
while (1):
    if (flag_data["dj_has_die"] == False):

        # Tell DJ I have the die and I'm waiting for him
        flag_data['bill_has_die'] = True
        flag_data['bill_waiting'] = False
        message = json.dumps(flag_data)
        client.publish("flag_data", message, qos=1)

        time.sleep(3)

        # go to ABOVE pickup die
        crx10.set_pose(pose5)
        crx10.start_robot()
        # Go drop die off
```

```
            crx10.set_pose(pose4)
            crx10.start_robot()

            break
        else:
            print("Waiting for DJ to release")
            print("DJ flag data waiting:", flag_data["dj_waiting"], " die:", flag_data["dj_has_die"])
            time.sleep(.5)

    # Open Gripper, drop die in square
    crx10.onRobot_gripper_close(90, 20)

    # go to ABOVE pickup die
    crx10.set_pose(pose5)
    crx10.start_robot()

    # go home
    crx10.set_pose(pose1)
    crx10.start_robot()

    client.loop_stop()
    client.disconnect()


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

5 Block Diagram

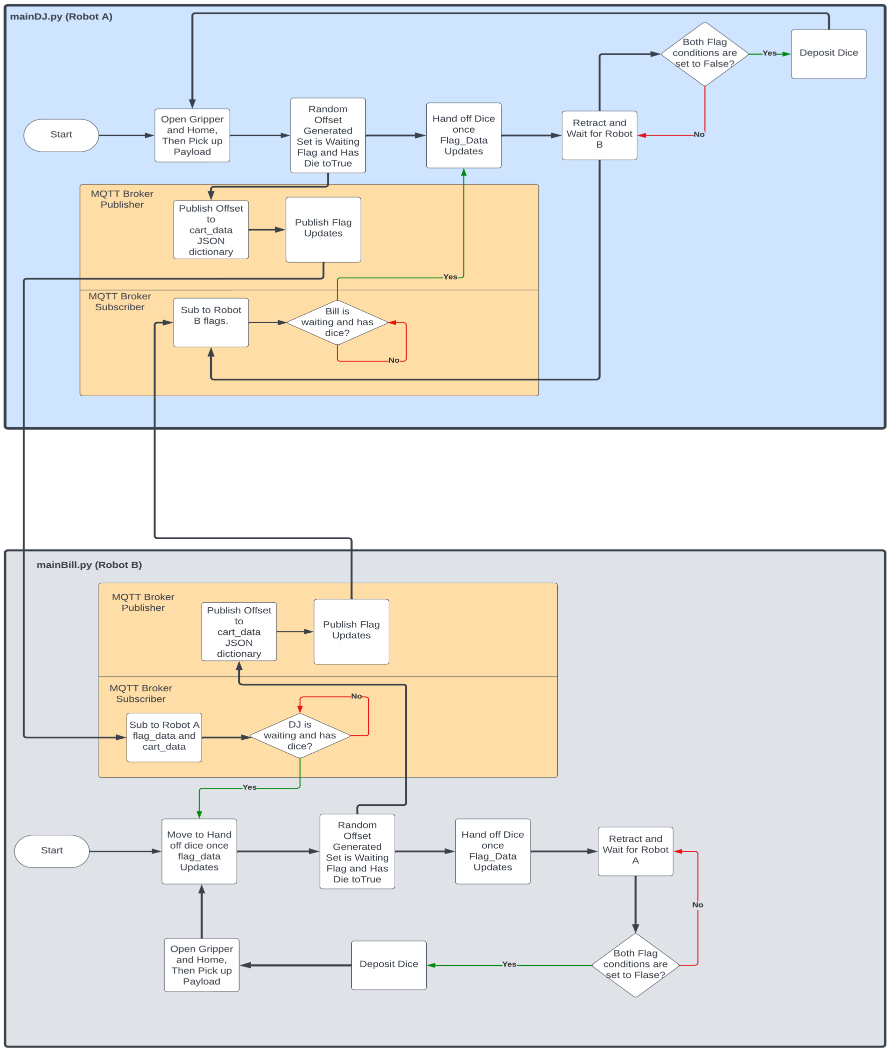
mainDJ.py (Robot A)
Start
Open Gripper and Home, Then Pick up Payload
Random Offset Generated Set is Waiting Flag and Has Die toTrue
Hand off Dice once Flag_Data Updates
Retract and Wait for Robot B
Both Flag conditions are set to False?
Yes
No
Deposit Dice
MQTT Broker Publisher
Publish Offset to cart_data JSON dictionary
Publish Flag Updates
MQTT Broker Subscriber
Sub to Robot B flags.
Bill is waiting and has dice?
Yes
No
mainBill.py (Robot B)
MQTT Broker Publisher
Publish Offset to cart_data JSON dictionary
Publish Flag Updates
MQTT Broker Subscriber
Sub to Robot A flag_data and cart_data
DJ is waiting and has dice?
No
Yes
Start
Move to Hand off dice once flag_data Updates
Random Offset Generated Set is Waiting Flag and Has Die toTrue
Hand off Dice once Flag_Data Updates
Retract and Wait for Robot A
No
Both Flag conditions are set to Flase?
Yes
Deposit Dice
Open Gripper and Home, Then Pick up Payload

Figure 1: Robot A and Robot B MQTT communications