

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)

```
1
2 # Advanced Robotics 2(CS 553)
3 # Author: Dan Blanchette
4 # Date: 10/23/23
5 # Lab 4: Dice Passing Robots
6 # Ver: 2.0
7
8 # Description: This program will use MQTT to communicate cartesian
9 # offsets and robot "states"
10 # to my partner Gary Bank's program. We use Boolean flags to
11 # indicate feedback to the other robot
12 # for grabbing the dice and when each robot arrives at a random
13 # walk position. The dice are then passed
14 # to his robot, which will do its random walk handoff and send me
15 # the new coordinate offset via the MQTT
16 # broker. # This program utilizes the University of Idaho Fanuc
17 # Python API driver, Paho MQTT, and Mosquitto
18 # acting as a local broker on Gary's laptop.
19
20 import os
21 import random
22 from robot_controller import robot
23 import paho.mqtt.client as mqtt
24 import json
25 import random
26 import time
27
28 # MQTT server details
29 BROKER_IP = "129.101.98.194"
30 BROKER_PORT = 1883
31 # DJ IP ADDRESS
32 drive_path = '129.101.98.215' # DJ
33
34 # Dictionary for Robot Hand off Start Location Pose
35 def_cart_data = {
36     "x": 364.646,
37     "y": 690.701,
38     "z": 376.777,
39     "w": -90.995,
40     "p": -31.562,
41     "r": -1.412
42 }
43
44 # Dictionary that will be used to update offset information for
45 # cartesian random walk
46 cart_data = {
47     "x": 0.0,
48     "y": 0.0,
49     "z": 0.0
50 }
51
52 flag_data = {
53     "dj_waiting": False,
54     "dj_has_dice": False,
```

```

50     "bill_waiting": False,
51     "bill_has_die": False
52 }
53
54 # robot API class instance
55 crx10_dj = robot(drive_path)
56
57 def on_publish(client, userdata, mid):
58     print(f'Message Published: {userdata}')
59
60
61 # Connect to broker verification
62 def on_connect(client, userdata, flags, rc):
63     print(f"Connected with result code {rc}")
64     client.subscribe("flag_data")
65     client.subscribe("cart_data")
66
67 # Disconnect from Broker
68 def on_disconnect(client, userdata, rc, properties=None):
69     print(f"Disconnected with result code {rc}")
70
71 # Topics to Subscribe to
72 def on_message(client, userdata, msg):
73     if msg.topic == "cart_data":
74         received_data = json.loads(msg.payload.decode())
75         cart_data['x'] = received_data.get('x', cart_data['x']) #
76         if first value DNE, grab second value
77         cart_data['y'] = received_data.get('y', cart_data['y'])
78         cart_data['z'] = received_data.get('z', cart_data['z'])
79
80     if msg.topic == "flag_data":
81         received_data = json.loads(msg.payload.decode())
82         flag_data['dj_waiting'] = received_data.get('dj_waiting',
83         flag_data['dj_waiting'])
84         flag_data['dj_has_die'] = received_data.get('dj_has_die',
85         flag_data['dj_has_die'])
86         flag_data['bill_waiting'] = received_data.get('bill_waiting',
87         flag_data['bill_waiting'])
88         flag_data['bill_has_die'] = received_data.get('bill_has_die',
89         flag_data['bill_has_die'])
90
91 # MQTT Client Setup
92 client = mqtt.Client()
93 client.on_publish = on_publish
94 client.on_connect = on_connect
95 client.on_disconnect = on_disconnect
96 client.on_message = on_message
97 client.connect(BROKER_IP, BROKER_PORT)
98 client.loop_start()
99
100 def main():
101     # main program
102     # Local vars that hold the home and payload approach joint poses

```

```

101     home =
102         [3.6055996417999268, -1.5429623126983643, 3.3683128356933594,
103         -0.713886559009552, -4.529087066650391, -2.439002752304077]
104     def_loc_grab = [17.481, 25.178, -51.212, 0.697, -38.636, 13.036]
105
106     # Start Robot Routine
107     # Open the Gripper
108     crx10_dj.shunk_gripper('open')
109     # Go to the home position
110     crx10_dj.write_joint_pose(home)
111     crx10_dj.start_robot()
112
113     # Move to pick up dice position
114     crx10_dj.write_joint_pose(def_loc_grab)
115     crx10_dj.start_robot()
116
117     # DEBUG TESTING VAR CHECK
118     # print(f'DJ is moving:{move_flag}')
119     # print(f'{moving}')
120
121     # Close the gripper (non-blocking)
122     crx10_dj.shunk_gripper('close')
123     # Update the has die flag and publish to topic for Gary to
124     # subscribe to
125     flag_data["dj_has_die"] = True
126     message = json.dumps(flag_data)
127     # publish grab flag as true and send to Gary's lappy
128     client.publish("flag_data", message, qos=1)
129     print(f'I just sent Gary This Value:{message}')
130
131     # Applying random offset to dictionary
132     cart_data["x"] = random.uniform(-50.0, 50.0)
133     print(f'x_off: {cart_data["x"]}')
134     cart_data["y"] = random.uniform(-50.0, 50.0)
135     print(f'y_off: {cart_data["y"]}')
136     cart_data["z"] = random.uniform(-90.0, 90.0)
137     print(f'z_off: {cart_data["z"]}')
138     # Sends offset data to Gary
139     message2 = json.dumps(cart_data)
140     client.publish("cart_data", message2, qos=2)
141
142     # Add the random offset to the default point of reference for
143     # the hand off
144     crx10_dj.write_cartesian_position(def_cart_data["x"] + cart_data
145     ["x"], def_cart_data["y"] + cart_data["y"], def_cart_data["z"]
146     + cart_data["z"],
147     def_cart_data["w"],
148     def_cart_data["p"], def_cart_data["r"])
149     # move to that position
150     crx10_dj.start_robot()
151
152     # Send Gary's computer the waiting to hand off flag set as True
153     flag_data["dj_waiting"] = True
154     message1 = json.dumps(flag_data)
155     print("DJ is waiting to hand off")
156     client.publish("flag_data", message1, qos=1)

```

```

152 # Poll Bill to see if robot has dice
153 while(1):
154     # Bill has the dice
155     if(flag_data["bill_has_die"] == True):
156         # DJ Opens gripper
157         crx10_dj.shunk_gripper("open")
158         # Reset DJ has dice flag to False
159         flag_data['dj_has_die'] = False
160         # Send Flag update to Gary's PC
161         message2 = json.dumps(flag_data)
162         client.publish("flag_data", message2, qos=1)
163
164         #if DJ has dice is false, ok to -500mm y-axis retract
165         if(flag_data["dj_has_die"]== False):
166             print("Retracting.....")
167             crx10_dj.write_cartesian_position(364.646, 190.701,
376.777, -90.995, -31.562, -1.412)
168             crx10_dj.start_robot()
169             time.sleep(0.2)
170             break
171         else:
172             # debug statements
173             print("waiting for Bill to grab")
174             print(f'Bill Status:{flag_data["bill_has_die"]}')
175             time.sleep(3)
176
177 # Poll to see if Bill has die and Bill is waiting
178
179 while(1):
180
181     if(flag_data["bill_has_die"] == True and flag_data["
bill_waiting"] == True):
182         # move y+ 10mm to grab dice
183         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"])
184         crx10_dj.start_robot()
185         # close the gripper
186         crx10_dj.shunk_gripper('close')
187         # set flags
188         flag_data["dj_has_die"] = True
189         flag_data["dj_waiting"] = False
190         # update gripper flag is closed and DJ has the dice
191         # and isn't waiting any more.
192         message3 = json.dumps(flag_data)
193         client.publish("flag_data", message3, qos=1)
194         break
195
196     else:
197         print("Waiting for Bill to let go")
198         print(f'Bill Dice Status:{flag_data["bill_has_die"]}')
199         print(f'Bill Waiting Status:{flag_data["bill_waiting"]}')
200     ')
201     time.sleep(3)
202
203 # wait for false flag from bill's gripper

```

```

204 while(1):
205     print(f'Bill has die == {flag_data["bill_has_die"]}')
206     if (flag_data["bill_has_die"] == False):
207         # ok to move to default pick up/drop off position
208         crx10_dj.write_joint_pose(def_loc_grab)
209         crx10_dj.start_robot()
210
211         # DICE CAN BE STICKY WHEN PLACING
212         # Wait 1.5 seconds after opening then go home
213         crx10_dj.shunk_gripper('open')
214         time.sleep(1.5)
215         crx10_dj.write_joint_pose(home)
216         crx10_dj.start_robot()
217         break
218
219
220 # REPETITION 2
221
222 # Go to the home position
223 crx10_dj.write_joint_pose(home)
224 crx10_dj.start_robot()
225
226 # Get the dice from the default
227 crx10_dj.write_joint_pose(def_loc_grab)
228 crx10_dj.start_robot()
229
230 # print statements for debugging flag data
231 # print(f'DJ is moving:{move_flag}')
232
233 # close gripper
234 crx10_dj.shunk_gripper('close')
235 # set DJ has die flag to True
236 flag_data["dj_has_die"] = True
237 # Convert to JSON string and Publish Flag Data to Broker
238 message = json.dumps(flag_data)
239 # publish grab flag as true and send to Gary's lappy
240 client.publish("flag_data", message, qos=1)
241 # Runtime Debugging Print Statement
242 print(f'I just sent Gary This Value:{message}')
243
244 # Applying random offset to dictionary
245 cart_data["x"] = random.uniform(-50.0, 50.0)
246 print(f'x_off: {cart_data["x"]}')
247 cart_data["y"] = random.uniform(-50.0, 50.0)
248 print(f'y_off: {cart_data["y"]}')
249 cart_data["z"] = random.uniform(-90.0, 90.0)
250 print(f'z_off: {cart_data["z"]}')
251 # Sends offset data to Gary
252 message2 = json.dumps(cart_data)
253 client.publish("cart_data", message2, qos=2)
254
255 # Add received offset from Bill and add to default reference
    values
256 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"],

```

```

257         def_cart_data["w"],
258         def_cart_data["p"], def_cart_data["r"])
259     crx10_dj.start_robot()
260
261     flag_data["dj_waiting"] = True
262     message1 = json.dumps(flag_data)
263     print("DJ is waiting to hand off")
264     client.publish("flag_data", message1, qos=1)
265
266     # Poll Bill to see if robot has dice
267     while(1):
268         # Bill has the dice
269         if(flag_data["bill_has_dice"] == True):
270             # DJ Opens gripper
271             crx10_dj.shunk_gripper("open")
272             # Reset DJ has dice flag to False
273             flag_data['dj_has_dice'] = False
274             # Send Flag update to Gary's PC
275             message2 = json.dumps(flag_data)
276             client.publish("flag_data", message2, qos=1)
277
278             #if DJ has dice is false, ok to -500mm y-axis retract
279             if(flag_data["dj_has_dice"]== False):
280                 # From current position, retract 500mm on y-axis
281                 crx10_dj.write_cartesian_position(def_cart_data["x"],
282                                                    def_cart_data["y"] - 500, def_cart_data["z"],
283                                                    def_cart_data["w"]
284                                                    ], def_cart_data["p"], def_cart_data["r"])
285                 crx10_dj.start_robot()
286                 time.sleep(0.5)
287                 print("Done Going Home Now.....")
288                 crx10_dj.write_joint_pose(home)
289                 crx10_dj.start_robot()
290                 break
291             else:
292                 # debug statements
293                 print("waiting for Bill to take dice")
294                 print(f'Bill Status:{flag_data["bill_has_dice"]}')
295                 time.sleep(3)
296
297 if __name__=="__main__":
298     main()
299 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)

```
1  # Gary Banks
2  # Robotics I
3  # Lab 4
4  #
5  #
6  #
7  # Command to start local mqtt broker
8  # /usr/local/sbin/mosquitto -c /Users/gary/Documents/code/
   robot/lab4/mosquitto.conf
9  #
10 #
11
12
13 import random
14 from robot_controller import robot
15 import paho.mqtt.client as mqtt
16 import time
17 import json
18 import FANUCethernetipDriver
19
20 # MQTT server details
21 BROKER_IP = "129.101.98.195"
22 BROKER_PORT = 1883
23
24 # This dictionary contains the cartesian offset from our handoff
   reference point.
25 # It will be updated with random values, and published to a "
   cart_data" MQTT topic
26 cart_data = {
27     "x": 0.0,
28     "y": 0.0,
29     "z": 0.0,
30 }
31
32 # This dictionary will contain the cartesian offset from our
   handoff reference point.
33 # It will be updated with random values, and published to a "
   cart_data" MQTT topic
34 flag_data = {
35     "dj_waiting": False,
36     "dj_has_die": False,
37     "bill_waiting": False,
38     "bill_has_die": False
39 }
40
```



```

41
42 def on_publish(client, userdata, mid):
43     print("Message Published...")
44
45
46 def on_connect(client, userdata, flags, rc):
47     print(f"Connected with result code {rc}")
48     client.subscribe("flag_data")
49     client.subscribe("cart_data")
50
51
52 def on_disconnect(client, userdata, rc, properties=None):
53     print(f"Disconnected with result code {rc}")
54
55
56 def on_message(client, userdata, msg):
57     if msg.topic == "cart_data":
58         print("Message received")
59
60         # DEBUGGING
61         # print("Received payload:", msg.payload.decode())
62
63         # Decode the JSON
64         received_data = json.loads(msg.payload.decode())
65
66         # Update cart Values
67         cart_data.update(received_data)
68
69         # DEBUGGING print types and values
70         print("FROM ON MESSAGE Cartesian values after Random x:",
71               cart_data["x"], "(", type(cart_data["x"]), ")",
72               " y:", cart_data["y"], "(", type(cart_data["y"]), ")",
73               ,
74               " z:", cart_data["z"], "(", type(cart_data["z"]), ")",
75               )
76
77     if msg.topic == "flag_data":
78         received_data = json.loads(msg.payload.decode())
79         flag_data['dj_waiting'] = received_data.get('dj_waiting',
80             flag_data['dj_waiting'])
81         flag_data['dj_has_die'] = received_data.get('dj_has_die',
82             flag_data['dj_has_die'])
83         flag_data['bill_waiting'] = received_data.get('bill_waiting',
84             flag_data['bill_waiting'])
85         flag_data['bill_has_die'] = received_data.get('bill_has_die',
86             flag_data['bill_has_die'])
87
88
89 # MQTT Setup
90 client = mqtt.Client()
91 client.on_publish = on_publish
92 client.on_connect = on_connect
93 client.on_disconnect = on_disconnect
94 client.on_message = on_message
95 client.connect(BROKER_IP, BROKER_PORT)
96
97 drive_path = '129.101.98.214' # CRX10 BILL

```

```

91 # Pose and cartesian position information
92 pose1 = [0.0, -2.409282387816347e-06, 0.009522347711026669,
          -0.024758676066994667, -0.018449142575263977,
93          0.0247572660446167] # Home position
94 pose2 = [-9.638092994689941, -9.305192947387695,
          -11.088014602661133, -1.9604847431182861, -78.9268798828125,
95          -80.95472717285156] # Approach Handoff
96 pose3 = [-38.522300720214844, 6.566395282745361,
          -11.665398597717285, -1.7110475301742554, -79.27543640136719,
97          -52.11516189575195] # Handoff
98 pose4 = [22.032983779907227, 30.85733985900879, -84.4652328491211,
          -21.550495147705078, 88.06071472167969,
99          -177.08120727539062] # Die Drop-off
100 pose5 = [21.896, 17.596, -68.232, -22.433, 72.983, -170.967] #
          ABOVE Die Drop-off
101
102 handoff = [400.062, -491.382, 444.861, -179.717, 1.903, -90.965] #
          cartesian
103
104 def main():
105     """! Main program entry"""
106
107     # DEBUGGING
108     # print("Cartesian values x:", cart_data["x"], " y:", cart_data
109     # ["y"], " z:", cart_data["z"])
110
111     # MQTT stuff
112     client.loop_start()
113
114     # Create new robot object
115     crx10 = robot(drive_path)
116
117     # Set robot speed
118     crx10.set_speed(200)
119
120     # Open Gripper
121     crx10.onRobot_gripper_close(90, 20)
122
123     # #-----POSE ADJUST
124     #
125     # # Move arm lift die off belt
126     # crx10.set_pose(pose4)
127     # crx10.start_robot()
128     #
129     # exit()
130     # #-----POSE ADJUST
131
132     # Move arm HOME
133     crx10.set_pose(pose1)
134     crx10.start_robot()
135
136     # Move arm to HANDOFF APPROACH
137     crx10.set_pose(pose2)
138     crx10.start_robot()
139
140     # Loop at HANDOFF APPROACH, wait for DJ to be waiting and have
141     # die.

```

```

140 while (1):
141     if flag_data["dj_waiting"] == True and flag_data["
dj_has_die"] == True:
142         # PRINT DEBUGGING WITH TYPE INFO
143         print("FROM ON MESSAGE Cartesian values after Random x:
", cart_data["x"], "(", type(cart_data["x"]), ")",
144             " y:", cart_data["y"], "(", type(cart_data["y"]),
145             " z:", cart_data["z"], "(", type(cart_data["z"]),
146             ")")
147
148     # copy handoff cartesian, apply new cart_data from DJ
to it
149     temp_handoff = handoff.copy() # What the heck
150
151     # check cart_datas
152     temp_handoff[0] += cart_data["x"]
153     temp_handoff[1] += cart_data["y"]
154     temp_handoff[2] += cart_data["z"]
155
156     # Move robot to handoff+random, y -80
157     crx10.send_coords(temp_handoff[0], temp_handoff[1]+80,
temp_handoff[2], temp_handoff[3], temp_handoff[4],
temp_handoff[5])
158     crx10.start_robot()
159     # Move to complete handoff + random
160     crx10.send_coords(temp_handoff[0], temp_handoff[1],
temp_handoff[2], temp_handoff[3], temp_handoff[4],
temp_handoff[5])
161     crx10.start_robot()
162     break
163 else:
164     print("Waiting for DJ")
165     print("DJ flag data waiting:", flag_data["dj_waiting"],
" die:", flag_data["dj_has_die"])
166     time.sleep(1)
167
168 # grasp die
169 crx10.onRobot_gripper_close(77, 15)
170 # Wait for grasp NEEDS TO BE OPTIMIZED
171 time.sleep(4)
172 # Tell DJ I have the die and I'm waiting for him
173 flag_data['bill_has_die'] = True
174 flag_data['bill_waiting'] = True
175 message = json.dumps(flag_data)
176 client.publish("flag_data", message, qos=1)
177
178 # Goto approach handoff, wait for DJ to release die
179 while 1:
180     if flag_data["dj_has_die"] == False:
181
182         # Tell DJ I have the die and I'm waiting for him
183         flag_data['bill_has_die'] = True
184         flag_data['bill_waiting'] = False
185         message = json.dumps(flag_data)
186         client.publish("flag_data", message, qos=1)
187
188

```

```

189         time.sleep(5)
190
191         # go to ABOVE pickup die
192         crx10.set_pose(pose5)
193         crx10.start_robot()
194         # Go drop die off
195         crx10.set_pose(pose4)
196         crx10.start_robot()
197
198         break
199     else:
200         print("Waiting for DJ to release")
201         print("DJ flag data waiting:", flag_data["dj_waiting"],
202               " die:", flag_data["dj_has_die"])
203         time.sleep(.5)
204
205     # Open Gripper, drop die in square
206     crx10.onRobot_gripper_close(90, 20)
207
208     # go to ABOVE pickup die
209     crx10.set_pose(pose5)
210     crx10.start_robot()
211
212     # go home
213     crx10.set_pose(pose1)
214     crx10.start_robot()
215
216     # Generate a random value between -50 and 50 for x and y, 100
217     for z
218         cart_data['x'] = round(random.uniform(-50.0, 50.0), 3)
219         cart_data['y'] = round(random.uniform(-50.0, 50.0), 3)
220         cart_data['z'] = round(random.uniform(-90.0, 100.0), 3)
221
222     # Publish Random offset
223     cart_message = json.dumps(cart_data)
224     client.publish("cart_data", cart_message, qos=1)
225
226     # Reset temp_handoff
227     temp_handoff = handoff.copy() # What the heck
228     # Apply random
229     temp_handoff[0] += cart_data['x']
230     temp_handoff[1] += cart_data['y']
231     temp_handoff[2] += cart_data['z']
232
233     # go to ABOVE pickup die
234     crx10.set_pose(pose5)
235     crx10.start_robot()
236
237     # go to pickup die
238     crx10.set_pose(pose4)
239     crx10.start_robot()
240
241     # CloseGripper
242     crx10.onRobot_gripper_close(77, 15)
243
244     # go approach handoff
245     crx10.set_pose(pose2)

```

```

244     crx10.start_robot()
245
246     # Move robot to handoff+random, y - 80
247     crx10.send_coords(temp_handoff[0], temp_handoff[1] + 80,
248                       temp_handoff[2], temp_handoff[3], temp_handoff[4],
249                       temp_handoff[5])
250     crx10.start_robot()
251     # Move to complete handoff + random
252     crx10.send_coords(temp_handoff[0], temp_handoff[1],
253                       temp_handoff[2], temp_handoff[3], temp_handoff[4],
254                       temp_handoff[5])
255     crx10.start_robot()
256
257     flag_data['bill_has_die'] = True
258     flag_data['bill_waiting'] = True
259     message = json.dumps(flag_data)
260     client.publish("flag_data", message, qos=1)
261
262     # Wait for DJ to grasp, release
263     while (1):
264         if (flag_data["dj_has_die"] == True):
265             # OpenGripper
266             crx10.onRobot_gripper_close(90, 20)
267             time.sleep(3)
268             # Tell DJ I have the die and I'm waiting for him
269             flag_data['bill_has_die'] = False
270             flag_data['bill_waiting'] = False
271             message = json.dumps(flag_data)
272             client.publish("flag_data", message, qos=1)
273
274             break
275         else:
276             print("Waiting for DJ to grasp")
277             print("DJ flag data waiting:", flag_data["dj_waiting"],
278                   " die:", flag_data["dj_has_die"])
279             time.sleep(.5)
280
281     # Move robot to handoff+random, y - 80
282     crx10.send_coords(temp_handoff[0], temp_handoff[1] + 80,
283                       temp_handoff[2], temp_handoff[3], temp_handoff[4],
284                       temp_handoff[5])
285     crx10.start_robot()
286
287     # Move arm to handoff approach
288     crx10.set_pose(pose2)
289     crx10.start_robot()
290
291     # -----END OF LOOP !!!!!
292
293     # Goto approach handoff, wait for DJ
294     while (1):
295         if flag_data["dj_waiting"] == True and flag_data["
296         dj_has_die"] == True:
297             # PRINT DEBUGGING WITH TYPE INFO
298             print("FROM ON MESSAGE Cartesian values after Random x:
299             ", cart_data["x"], "(", type(cart_data["x"]), ")")

```

```

294         " y:", cart_data["y"], "(", type(cart_data["y"]),
295         ")",
296         " z:", cart_data["z"], "(", type(cart_data["z"]),
297         ")",
298
299     # copy handoff cartesian, apply new cart_data from DJ
300     to it
301     temp_handoff = handoff.copy() # What the heck
302
303     # check cart_datas
304     temp_handoff[0] += cart_data["x"]
305     temp_handoff[1] += cart_data["y"]
306     temp_handoff[2] += cart_data["z"]
307
308     # Move robot to handoff+random, y - 80
309     crx10.send_coords(temp_handoff[0], temp_handoff[1] +
310     80, temp_handoff[2], temp_handoff[3], temp_handoff[4],
311     temp_handoff[5])
312     crx10.start_robot()
313     # Move to complete handoff + random
314     crx10.send_coords(temp_handoff[0], temp_handoff[1],
315     temp_handoff[2], temp_handoff[3], temp_handoff[4],
316     temp_handoff[5])
317     crx10.start_robot()
318     break
319     else:
320         print("Waiting for DJ")
321         print("DJ flag data waiting:", flag_data["dj_waiting"],
322         " die:", flag_data["dj_has_die"])
323         time.sleep(1)
324
325     # #grasp die
326     crx10.onRobot_gripper_close(77, 15)
327     # Wait for grasp
328     time.sleep(3)
329     # Tell DJ I have the die and I'm waiting for him
330     flag_data['bill_has_die'] = True
331     flag_data['bill_waiting'] = True
332     message = json.dumps(flag_data)
333     client.publish("flag_data", message, qos=1)
334
335     # Goto approach handoff, wait for DJ
336     while (1):
337         if (flag_data["dj_has_die"] == False):
338
339             # Tell DJ I have the die and I'm waiting for him
340             flag_data['bill_has_die'] = True
341             flag_data['bill_waiting'] = False
342             message = json.dumps(flag_data)
343             client.publish("flag_data", message, qos=1)
344
345             time.sleep(3)
346
347             # go to ABOVE pickup die
348             crx10.set_pose(pose5)
349             crx10.start_robot()
350             # Go drop die off

```

```

345         crx10.set_pose(pose4)
346         crx10.start_robot()
347
348         break
349     else:
350         print("Waiting for DJ to release")
351         print("DJ flag data waiting:", flag_data["dj_waiting"],
352               " die:", flag_data["dj_has_die"])
353         time.sleep(.5)
354
355     # Open Gripper, drop die in square
356     crx10.onRobot_gripper_close(90, 20)
357
358     # go to ABOVE pickup die
359     crx10.set_pose(pose5)
360     crx10.start_robot()
361
362     # go home
363     crx10.set_pose(pose1)
364     crx10.start_robot()
365
366     client.loop_stop()
367     client.disconnect()
368
369 if __name__ == "__main__":
370     main()

```

5 Block Diagram

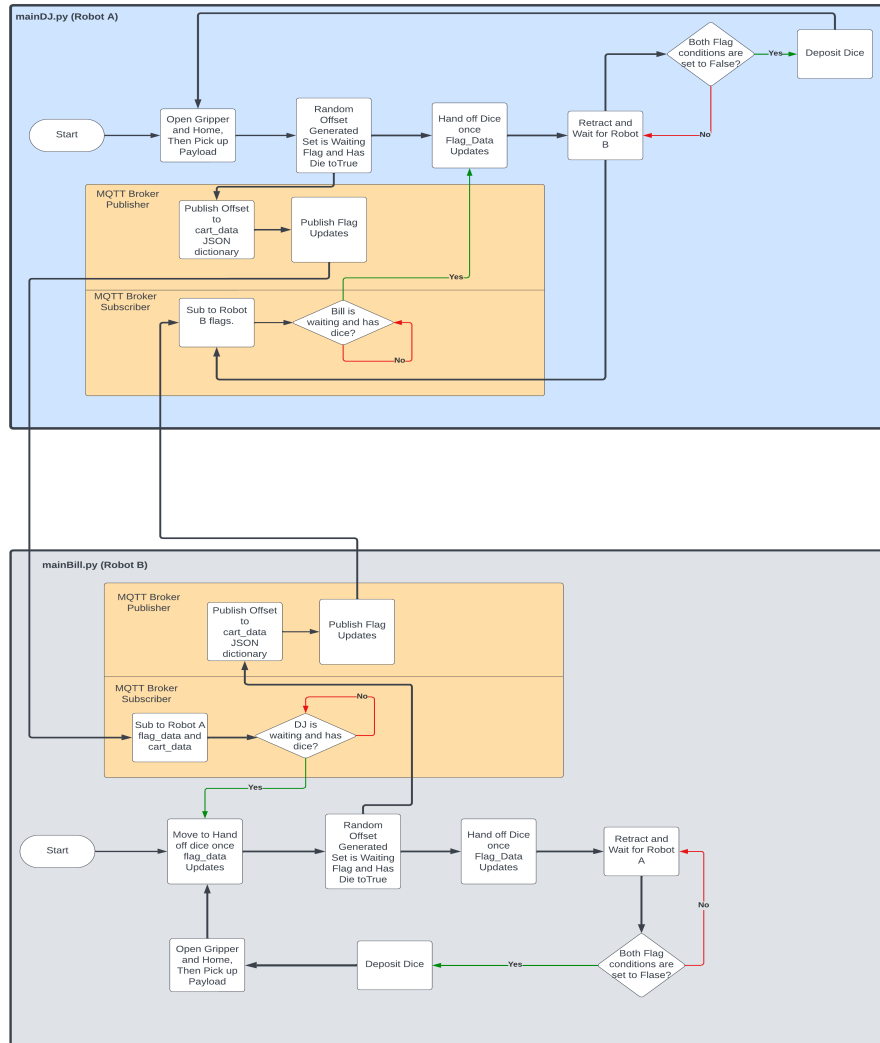


Figure 1: Robot A and Robot B MQTT communications