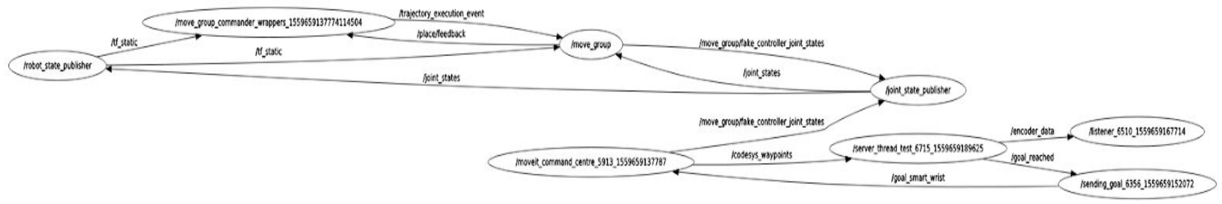
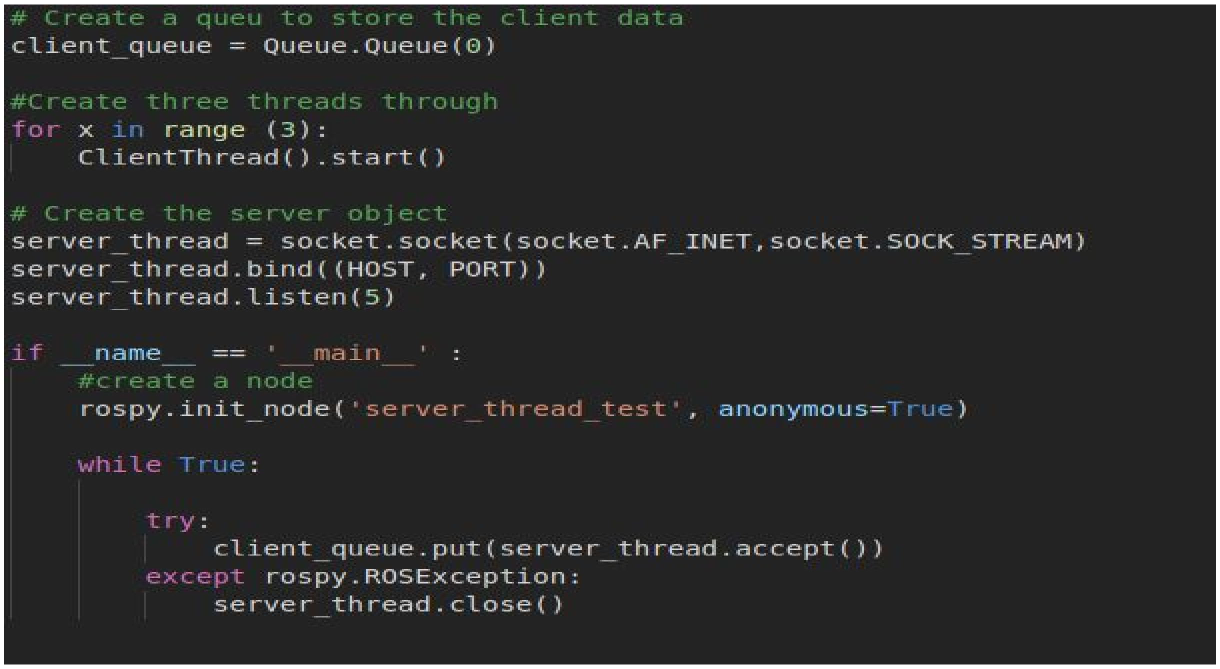
ROS – CodeSys Communication

ROS


/move_group_commander_wrappers_1559659137774114504
/trajectory_execution_event
/place/feedback
/tf_static
/tf_static
/robot_state_publisher
/move_group
/joint_states
/move_group/fake_controller_joint_states
/joint_states
/joint_state_publisher
/move_group/fake_controller_joint_states
/encoder_data
/listener_6510_1559659167714
/server_thread_test_6715_1559659189625
/goal_reached
/moveit_command_centre_5913_1559659137787
/codesys_waypoints
/goal_smart_wrist
/sending_goal_6356_1559659152072

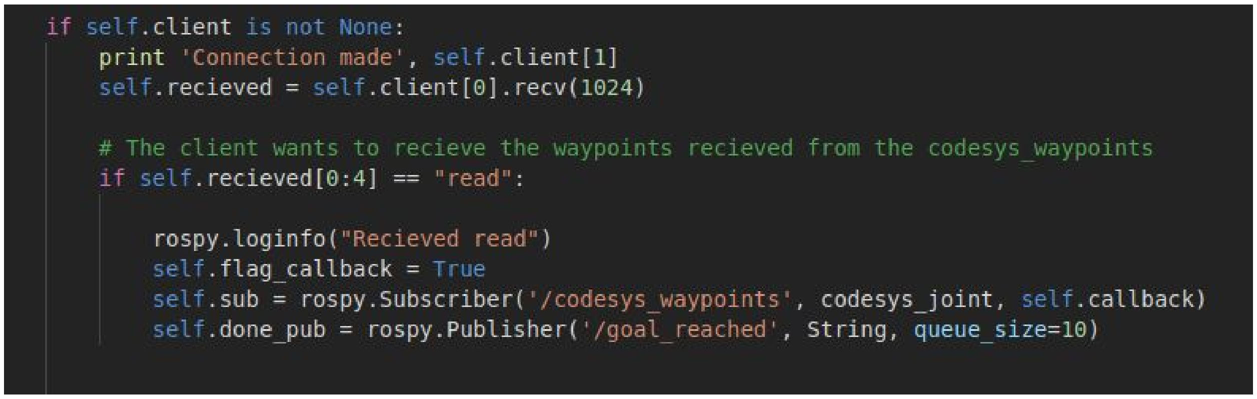
server_thread_test is the node responsible for creating multithreaded servers that will connect to the Codesys through the TCP/IP socket.

When a connection is made a server is created on the same process to handle the client



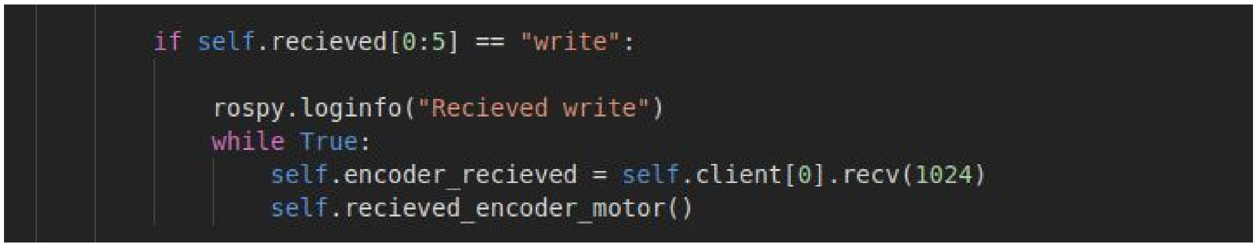
```
# Create a queu to store the client data
client_queue = Queue.Queue(0)

#Create three threads through
for x in range (3):
    ClientThread().start()

# Create the server object
server_thread = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
server_thread.bind((HOST, PORT))
server_thread.listen(5)

if __name__ == '__main__' :
    #create a node
    rospy.init_node('server_thread_test', anonymous=True)

    while True:

        try:
            client_queue.put(server_thread.accept())
        except rospy.ROSException:
            server_thread.close()
```

I limited the number of threads to 3, if there is more than 3 clients trying to connect, their connection will be put in queue and handled by one of the three servers when it's done with its current client.

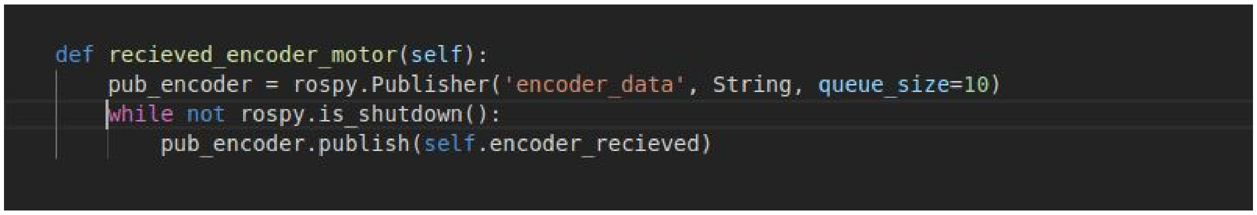


```python
if self.client is not None:
    print 'Connection made', self.client[1]
    self.recieved = self.client[0].recv(1024)

    # The client wants to recieve the waypoints recieved from the codesys_waypoints
    if self.recieved[0:4] == "read":

        rospy.loginfo("Recieved read")
        self.flag_callback = True
        self.sub = rospy.Subscriber('/codesys_waypoints', codesys_joint, self.callback)
        self.done_pub = rospy.Publisher('/goal_reached', String, queue_size=10)
```

One server thread subscribes to /codesys_waypoints to read the waypoints extracted from end-effector position by moveit when received by Codesys client “read”



```python
if self.recieved[0:5] == "write":

    rospy.loginfo("Recieved write")
    while True:
        self.encoder_recieved = self.client[0].recv(1024)
        self.recieved_encoder_motor()
```



```python
def recieved_encoder_motor(self):
    pub_encoder = rospy.Publisher('encoder_data', String, queue_size=10)
    while not rospy.is_shutdown():
        pub_encoder.publish(self.encoder_recieved)
```

One server thread publishes to the /goal_reachded topic when received “write” from Codesys client, where it published the encoder data coming from the motors tp be simulated by rviz

CodeSys

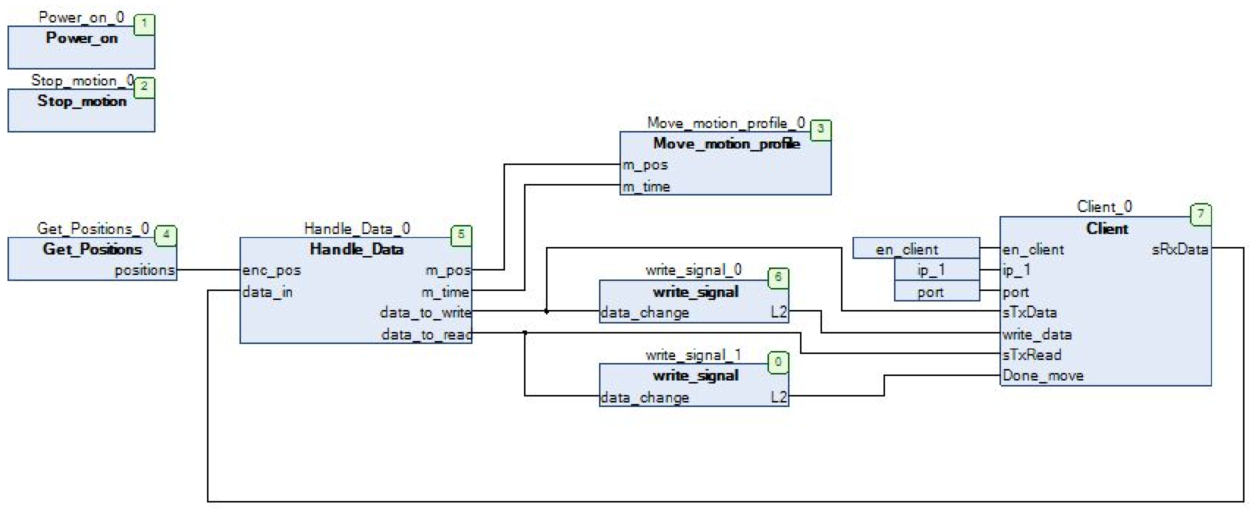


*Figure SEQ Figure * ARABIC 1 Main Program*

On the CodeSys side, we start the communication by enabling the client through a Boolean value “en_client”, this will initiate a connection with the server of “ip_1” through the given port. Inside the client there are two clients used, one for reading the incoming data and sending a done signal when the movement is over, and the other is for updating ROS with the current encoder positions.

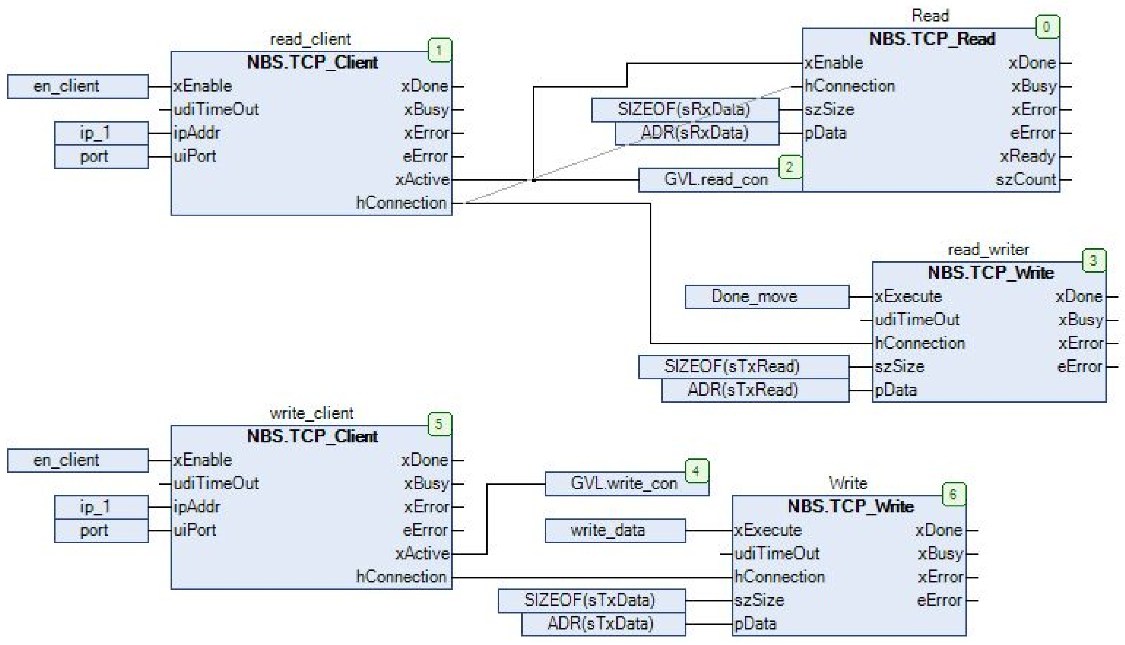


*Figure SEQ Figure * ARABIC 2Client*

Then the received data is fed into the “Handle_Data” block which changes the data from a string to arrays of floats. These arrays contain the setpoints of the 4 motors and the time intervals between the waypoints. They are then passed on to “Move_motion_profile” where every motor is actuated through its own block as shown in figure 3.

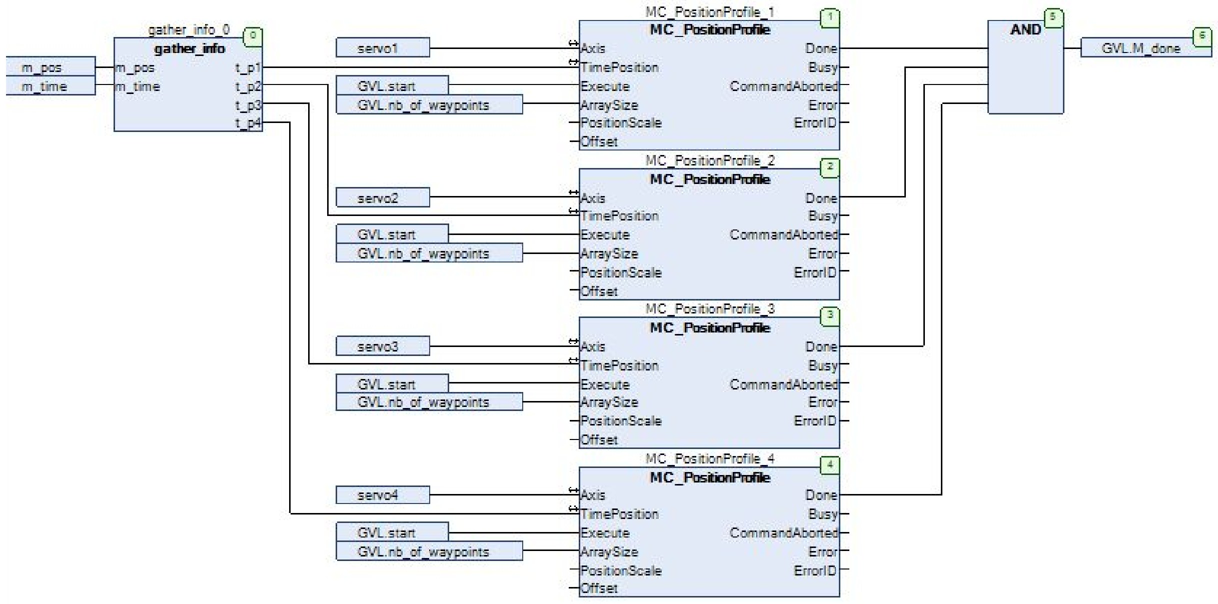


Figure 3Move_motion_profile

At the same time, the “Get_positions” block reads the data from the encoders and feeds them back to the client. The client then executes the writing block with “write_signal”, a block that generates a pulse once the data to be written changes.