Individual Lab Report 5 - Progress Review 4

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Anthony Kyu

Team C:

Kaushik Balasundar | Parker Hill | Anthony Kyu
Sundaram Seivur | Gunjan Sethi

April 7th, 2022

Carnegie Mellon University
The Robotics Institute

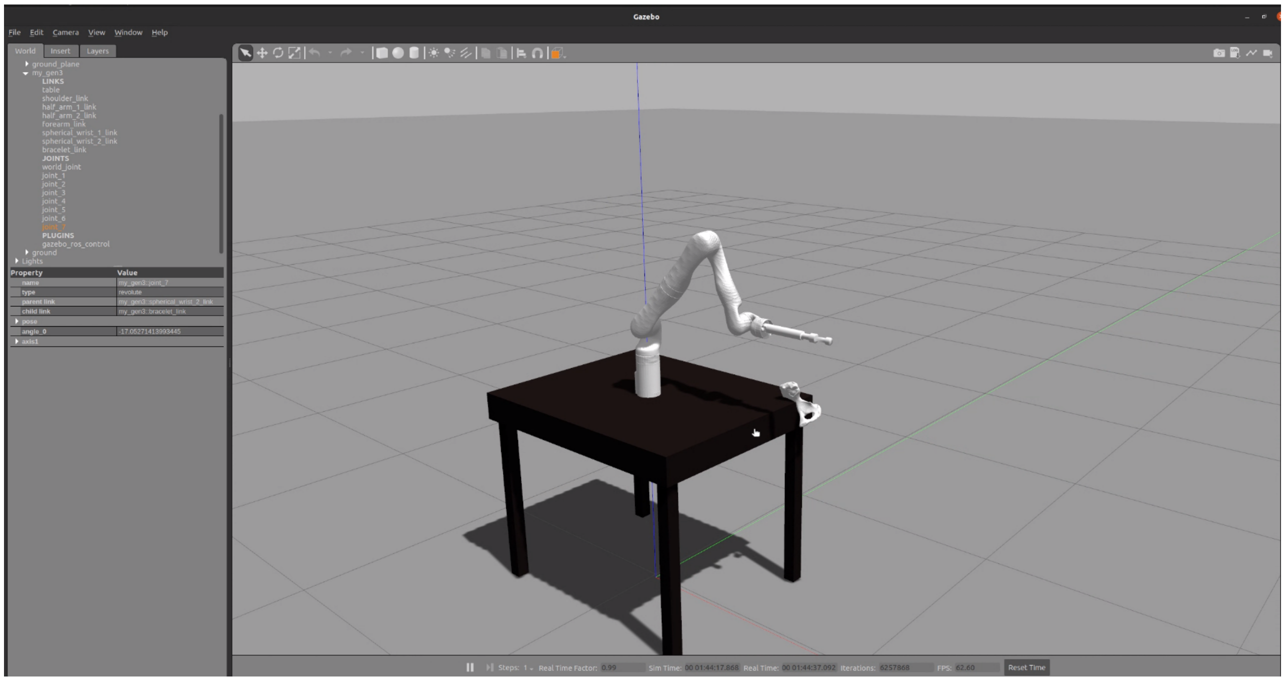
Contents

1 Individual Progress

Since the last progress review (PR3), I was responsible for taking the Model Predictive Controller Code that I had written in the last progress review and further optimize it, integrate it with ROS and test it in Gazebo (real-time) simulation. There was also a goal to implement the other states (Cartesian pose, Cartesian velocities, and end-effector forces), but was never accomplished due to challenges discussed later.

The first week was spent translating the Jupyter notebook I had written as an MPC into an executable Julia script for ROS to use. I then integrated the script into ROS, writing a ROS Julia node using RobotOS.jl, writing an MPC node and a Simulation node to do some initial testing to see how the MPC would handle in real-time simulation, using RK4 to simulate the dynamics in the Simulation node. The MPC worked decently well in this setup but, the success of the MPC highly relied on consistent CPU runtime and short solver convergence times, which wasn't always achieved.

However, since the simulation node could only run at a limited rate, I spent the second week further integrating the MPC into ROS, specifically integrating it with the Pilz Motion Planner nodes (written by Sundaram), and the Gazebo effort controller nodes (written by Kaushik). Once integrated with the Gazebo simulation (Figure 1), the MPC was further tested and tuned, but could not perform well enough to demonstrate for this progress review. The challenges with testing and tuning will be discussed in the challenges section.



Property	Value
name	my_gen3::joint_7
type	revolute
parent link	my_gen3::spherical_wrist_2_link
child link	my_gen3::bracelet_link
pose	
angle_0	-17.05271413993445
axis1	

```
Steps: 1  Real Time Factor: 0.99  Sim Time: 00 01:44:17.868  Real Time: 00 01:44:37.092  Iterations: 6257868  FPS: 62.60
```

Figure 1: Real-Time Simulation of Tracking PD Impedance Controller Moving Through a Straight-Line Trajectory in Gazebo

Because the MPC wasn't performing well, and our team wants to demonstrate our hardware for SVD, I also worked on developing a simple Tracking PD Impedance controller, which moves from waypoint to waypoint. This seemed to perform qualitatively well for tracking a trajectory (Figure 1), but more quantitative tests will be performed in the coming weeks. We will be developing both controllers in parallel, planning to first implement PD control on hardware, and then further develop the MPC to implement later (most likely in the Fall).

MPC and Controller ROS nodes are shown in Appendix A and B.

2 Challenges

The major challenges for this progress review were optimizing and tuning the MPC to have fast convergence times, and getting it to be robust in real-time simulation. As shown in the last progress review, the MPC performed very well in offline simulation, where the given time to solve an optimal control problem between discrete time steps is infinite. However, this is not a luxury given when doing real-time control, which means optimization needed to be done to get similar results to offline simulation. To meet this challenge, several methods were used to decrease convergence time, which included decreasing the time horizon of the MPC, and using different objective functions (LQR objectives vs tracking objectives). However, while these did improve the convergence time, it did not improve it enough to make a robust MPC controller. Furthermore, given that most of the libraries we were using were already well optimized in Julia, there seemed to be a bottleneck that would require us to swap our code over to another language such as C++ and use similar but different libraries, further delaying progress.

Another challenge in getting the MPC to properly work in Gazebo was that there was a slight model mismatch between the model we used to simulate in Gazebo to the model used in the MPC. This challenge was mitigated by checking URDF files to be consistent, but there seems to still be a small mismatch. After mitigation, the performance of both the PD controller and MPC controller drastically increased, but the MPC controller still did not converge fast enough. This issue was not discovered until too close to the progress review, and will be addressed the next two weeks.

The challenges mentioned above unfortunately caused delays in my progress, so I was unable to implement the other states into the MPC as originally planned. Luckily our team has built plenty of time in our schedule for such delays if needed.

3 Team Work

3.1 Anthony Kyu

Anthony worked on transferring the MPC code from offline simulation to online real-time simulation in Gazebo. This task involved writing MPC update functions as well as restructuring the code to be modular and more efficient. He then integrated the MPC code into ROS using RobotOS.jl, writing an MPC solver node to solve the optimal control problem, a simulation node for internal testing before integration, and a controller node to send torques to the effort controllers. Collaborating with Sundaram and Kaushik, Anthony then integrated the controller and MPC nodes with the trajectory planning nodes, Gazebo sensor nodes, and the Gazebo effort controllers, enabling real-time simulation testing of the controllers in Gazebo. Because the MPC had trouble performing well in real-time, Anthony also developed a PD Impedance Tracking controller in parallel to use as a fallback should the MPC not be viable by the SVD. Anthony also collaborated with Parker, providing tips on how to set up and wire the power distribution system.

3.2 Parker Hill

Parker redesigned the end-effector reamer to be shorter and more robust than his initial prototype. These designs were then 3D-printed and attached to the arm, allowing for the hardware system to be finalized for the Spring Validation Demonstration. He also worked on the motor control PCB, ordering all the parts, soldering them to the PCB, and testing the PCB for efficacy. While the PCB is not finalized, he was able to extend the motor wires and connect them to a power supply to actuate the reamer head, allowing for testing of the hardware system to be conducted on the sawbone pelvis.

3.3 Sundaram Seivur

Sundaram worked on completing the motion planning pipeline for which he compiled a custom message type on ROS. In this message he compiled trajectory messages which stored the joint states, a pose array message which stored the Cartesian positions and orientations, and a point array message which stored the Cartesian velocity. He wrote a node to plan a trajectory using the Pilz motion planning pipeline and publish the generated trajectory to a topic. He collaborated with Anthony and Kaushik for integrating the planning and controls subsystems. He also collaborated with Parker to ideate hardware designs and their interaction.

3.4 Kaushik Balasundar

Kaushik worked with Gunjan to obtain the registration marker's tip pose using the market geometry and the pose of the probe center obtained from the camera. He then worked with her to obtain the pointcloud of the pelvis using this probe, and drafted a software architecture for the perception sub-system. After this, he worked on using the acquired pointcloud to develop a method to obtain the initial guess for registration, and further refined the pipeline to register the pointcloud of the acetabulum with the 3D CAD model of the pelvis. The registration was then evaluated quantitatively. He also assisted Sundaram and Anthony in the integration of planning and controls sub-systems by writing the effort commander interface. Kaushik and Gunjan also discussed a revised software architecture for the perception and sensing subsystem.

3.5 Gunjan Sethi

Gunjan worked on extending the functionality of the perception pipeline and testing for robustness and reliability. She worked closely with Kaushik to integrate the new registration probe into the current pointcloud collection pipeline, obtain the probe tip pose and publish pointclouds at several frequencies. She developed user-input based sparse pointcloud collection and continuous dense pointcloud collection. Both functionalities were thoroughly tested. The point cloud collection pipeline was integrated and tested with the registration pipeline with the help of Kaushik. Further, she wrote test scripts to track multiple marker geometries as a proof-of-concept. Kaushik and Gunjan also discussed a revised software architecture for the perception and sensing subsystem.

4 Plans

For the next progress review, I plan to primarily focus on getting the Tracking PD Impedance controller working on the real hardware system, and preparing for the SVD, collaborating with Sundaram to get the Kinova Gen 3's wrench controllers working, and testing the PD controller. This will likely require more tuning the PD controller and also writing ROS nodes to directly get joint encoder readings to get joint positions and velocities. We would also like to get measurements of our force/torque sensor, so setting that up and integrating it with our system is another task to do.

If time permits, I will work more on the MPC as a stretch goal, deciding on future approaches to take, whether that is further optimizing it in Julia or completely switching to C++. Our team has already identified libraries that could be used in C++.

5 Appendix A: Controller_Node.jl

Please note that the following code is currently being developed, and is not complete. This node determined which controller was being used (PD or MPC control) and sent torque commands to the Gazebo effort controllers based on which controller was picked. If MPC control was used, it would communicate with the MPC Node to solve the optimal control problem.

```
#!/usr/bin/env julia

include("MPC.jl")

# Black magic to make Julia and Python happy
using Distributed

using Pkg
Distributed.@everywhere using Pkg

Distributed.@everywhere begin
  ENV["PYTHON"] = "/usr/bin/python3"
  Pkg.build("PyCall")
end

using PyCall
Distributed.@everywhere using PyCall


#Import Statements for RobotOS
using RobotOS

@rosimport trajectory_msgs.msg: JointTrajectory, JointTrajectoryPoint
@rosimport arthur_planning.msg: arthur_traj
@rosimport sensor_msgs.msg: JointState
rostypegen()
using .trajectory_msgs.msg
using .arthur_planning.msg
using .sensor_msgs.msg

function callback(msg::JointTrajectory, X, U, initialized, N, idx, lastTime)
    if msg.header.stamp != lastTime[1]
        lastTime[1] = msg.header.stamp
        N .= length(msg.points)
        for k = 1:N[1]
            X[k][1:7] .= msg.points[k].positions
            X[k][8:14] .= msg.points[k].velocities
            U[k][1:end] .= msg.points[k].effort
        end
        if !initialized[1] && N[1] > 0
            initialized .= true
        end
        idx .= 1
    end
end

function traj_callback(msg::arthur_traj, Xref)
    Xref[1] = typeof(zeros(14))[]
    for k=1:length(msg.traj.points)
        q = msg.traj.points[k].positions
        q = msg.traj.points[k].velocities
        push!(Xref[1], [q; q])
    end
    # RobotOS.loginfo("In Callback")
    # println(Xref[1][1])
end

function callback_x0(msg::JointState, x0)
    x0_new = zeros(14)
    x0_new[1:7] .= msg.position
    x0_new[8:14] .= msg.velocity
    copy!(x0, x0_new)
end

function loop(sub_obj1, sub_obj2, sub_obj3, pub_obj, x0, X, U, params,
 ↪  initialized, N, Xref, mpc_controller, idx, starting)
    loop_rate = Rate(20)
    while ! is_shutdown()
        RobotOS._run_callbacks(sub_obj1)
        RobotOS._run_callbacks(sub_obj2)
        RobotOS._run_callbacks(sub_obj3)
        if initialized[1] && mpc_controller[1]
            i = argmin(norm.([(X[k] - x0) for k=1:N[1]]))
            if norm(x0[1:7] - Xref[1][end][1:7]) < 0.05
                mpc_controller .= false
            end
            # if !isnan(x0[1])
            #     RobotOS.loginfo("MPC Control")
            #     println(U[i])
            #     println(x0)
            #     println(norm(x0 - Xref[1][end]))
            # end
            JointTrajectoryOutput = JointTrajectoryPoint()
            JointTrajectoryOutput.effort = U[i]
            RobotOS.loginfo("MPC Control")
            println(i)
            println(U[i])
            println(x0)
            # JointTrajectoryOutput.positions = x0[1:7]
            # JointTrajectoryOutput.velocities = x0[8:14]
            publish(pub_obj, JointTrajectoryOutput)
            # idx[1] = min(N[1], idx[1] + 1)
        # elseif norm(x0[8:14]) < 1e-6
        #     set_configuration!(params.state, x0[1:7])
        #     set_velocity!(params.state, zeros(7))
        #     params.v .= zeros(7)
        #     U_grav = inverse_dynamics(params.state, params.v)
        #     RobotOS.loginfo("Gravity Compensation Control")
        #     println(U_grav)
        #     println(x0)
        #     # println(RobotDynamics.dynamics(params.model, x0, U_grav))
        #     JointTrajectoryOutput = JointTrajectoryPoint()
        #     JointTrajectoryOutput.effort = U_grav
        #     publish(pub_obj, JointTrajectoryOutput)
        elseif starting[1]
            if norm(x0 - Xref[1][1]) < 1e-2
                starting .= false
            end
            set_configuration!(params.state, x0[1:7])
            set_velocity!(params.state, x0[8:14])
            i = 1
            x = Xref[1][i][1:7] - x0[1:7]
            # println(x)
            Kp = [30, 20, 20, 20, 20, 50, 20]
            Kd = [20, 20, 20, 20, 20, 20, 20]
            params.v .= (Kp .* x) .- (Kd .* x0[8:14])
            U_PD = inverse_dynamics(params.state, params.v)
            RobotOS.loginfo("PD Control")
            println(U_PD)
            println(x0)
            println(norm(x0 - Xref[1][1]))
            JointTrajectoryOutput = JointTrajectoryPoint()
            JointTrajectoryOutput.effort = U_PD
            publish(pub_obj, JointTrajectoryOutput)
        else
            set_configuration!(params.state, x0[1:7])
            set_velocity!(params.state, x0[8:14])
```

```
            # if initialized[1]
            #     i = argmin(norm.([(Xref[1][k][1:7] - x0[1:7]) for
             ↪  k=1:length(Xref[1])]))
            # else
            #     i = 1
            #     # println(Xref[1][i][1:7])
            #     # println(x0[1:7])
            # end

            i = min(argmin(norm.([(Xref[1][k][1:7] - x0[1:7]) for
             ↪  k=1:length(Xref[1])])) + 1, length(Xref[1]))

            x = Xref[1][i][1:7] - x0[1:7]
            # println(x)
            Kp = [100, 100, 100, 100, 100, 100, 100]
            Kd = [20, 20, 20, 20, 20, 20, 20]
            params.v .= (Kp .* x) .- (Kd .* x0[8:14])
            U_PD = inverse_dynamics(params.state, params.v)
            # x0 .= rk4(params.model, x0, U_PD, params.dt)
            # if !isnan(x0[1])
            #     RobotOS.loginfo("PD Control")
            #     println(params.v[1:7])
            #     println(U_PD[1:7])
            #     println(x0[1:7])
            #     println(norm(x0 - Xref[end]))
            # end
            RobotOS.loginfo("PD Tracking Control")
            println(U_PD)
            println(x0)
            println(i)
            JointTrajectoryOutput = JointTrajectoryPoint()
            JointTrajectoryOutput.effort = U_PD
            publish(pub_obj, JointTrajectoryOutput)
        end
        rossleep(loop_rate)
    end
end

function main()
    init_node("Controller_Node")
    params = MPC_Params()

    # Xref = get_trajectory(params)
    # x0 = copy(Xref[1])
    Xref = [typeof(zeros(14))[]]
    x0 = zeros(14)
    # N = [params.H-1]
    N = [params.LQRH]
    X = [zeros(14) for k = 1:params.H-1]
    U = [zeros(7) for k = 1:params.H-1]
    initialized = [false]
    mpc_controller = [false]
    starting = [true]
    idx = [1]
    lastTime = [RobotOS.now()]

    sub_x0 = Subscriber{JointState}("/my_gen3/joint_states", callback_x0,
     ↪  (x0,), queue_size=1)
    sub = Subscriber{JointTrajectory}("joint_torques",callback,(X, U,
     ↪  initialized, N, idx, lastTime),queue_size=1)
    sub_traj =
     ↪  Subscriber{arthur_traj}("/my_gen3/arthur_traj",traj_callback,(Xref,),queue_size=

    pub = Publisher{JointTrajectoryPoint}("mpc_torques",queue_size=1)

    while length(Xref[1]) == 0
        RobotOS._run_callbacks(sub_traj)
    end
    while norm(x0 - zeros(14)) == 0
        RobotOS._run_callbacks(sub_x0)
    end

    loop(sub, sub_traj, sub_x0, pub, x0, X, U, params, initialized, N, Xref,
     ↪  mpc_controller, idx, starting)
end

if !isinteractive()
    main()
end
```

6 Appendix B: MPC_Node.jl

Please note that the following code is currently being developed, and is not complete. This node communicated with the controller node and solved the optimal control problem, solving for the torques needed to follow the given trajectory while holding constraints.

```
#!/usr/bin/env julia

include("MPC.jl")

# Black magic to make Julia and Python happy
using Distributed

using Pkg
Distributed.@everywhere using Pkg

Distributed.@everywhere begin
  ENV["PYTHON"] = "/usr/bin/python3"
  Pkg.build("PyCall")
end

using PyCall
Distributed.@everywhere using PyCall


#Import Statements for RobotOS
using RobotOS

@rosimport trajectory_msgs.msg: JointTrajectory, JointTrajectoryPoint
@rosimport arthur_planning.msg: arthur_traj
@rosimport sensor_msgs.msg: JointState
rostypegen()
using .trajectory_msgs.msg
using .arthur_planning.msg
using .sensor_msgs.msg

# function get_trajectory(params)
#     qref = typeof(zeros(7))[]
#     push!(qref,  [4.9268856741946365e-05, 0.26016423071338934,
 ↪  3.140069344584098, -2.27008100955155, -2.40769461674617e-05,
 ↪  0.9598460188468598, 1.569995694393751])
#     push!(qref,  [-0.003768053224579125, 0.26079787208394173,
 ↪  3.1395327288847428, -2.269096150780195, -0.004224076946167462,
 ↪  0.9616197667116058, 1.5721591274301712])
```

```
#     push!(qref,  [-0.01522001946854234, 0.2626987961955988,
 ↪  3.137922881786677, -2.2661415744661304, -0.016824076946167464,
 ↪  0.9669410103058437, 1.578649426539432])
#     push!(qref,  [-0.03430662987514771, 0.26586700304836064,
 ↪  3.135239803289901, -2.2612172806093556, -0.03782407694616747,
 ↪  0.9758097496295736, 1.5894665917215332])
#     push!(qref,  [-0.0610278844443952, 0.2703024926422273,
 ↪  3.131483493394414, -2.254323269209871, -0.06722407694616747,
 ↪  0.9882259846827953, 1.6046106229764747])
#     push!(qref,  [-0.0953296828066516, 0.2759962847990445,
 ↪  3.126661557197997, -2.2454734980167035, -0.10496455313664366,
 ↪  1.0041645773154986, 1.6240508594052145])
#     push!(qref,  [-0.13259401741002397, 0.2821818315115795,
 ↪  3.121423165847148, -2.23585940048681, -0.14596455313664364,
 ↪  1.0214797350427807, 1.6451700866655072])
#     push!(qref,  [-0.1698583520133963, 0.2883673782241145,
 ↪  3.116184774496299, -2.226245302956917, -0.18696455313664362,
 ↪  1.0387948927700625, 1.6662893139258002])
#     push!(qref,  [-0.20712268661676866, 0.2945529249366496,
 ↪  3.1109463831454507, -2.2166312054270234, -0.2279645531366436,
 ↪  1.0561100504973449, 1.6874085411860928])
#     push!(qref,  [-0.24438702122014103, 0.3007384716491846,
 ↪  3.1057079917946018, -2.20701710789713, -0.2689645531366436,
 ↪  1.0734252082246267, 1.7085277684463855])
#     push!(qref,  [-0.2816513558235134, 0.3069240183617196,
 ↪  3.100469600443753, -2.1974030103672364, -0.3099645531366436,
 ↪  1.090740365951909, 1.7296469957066785])
#     push!(qref,  [-0.3189156904268857, 0.31310956507425464,
 ↪  3.095231209092904, -2.1877889128373433, -0.3509645531366436,
 ↪  1.108055523679191, 1.7507662229669712])
#     push!(qref,  [-0.3561800250302581, 0.3192951117867897,
 ↪  3.089992817742055, -2.17817481530745, -0.39196455313664363,
 ↪  1.1253706814064732, 1.7718854502272638])
#     push!(qref,  [-0.3934443596336305, 0.3254806584993247,
 ↪  3.0847544263912066, -2.1685607177775563, -0.43296455313664367,
 ↪  1.1426858391337553, 1.7930046774875568])
#     push!(qref,  [-0.43070869423700286, 0.33166620521185974,
 ↪  3.0795160350403576, -2.158946620247663, -0.47396455313664365,
 ↪  1.1600009968610374, 1.8141239047478495])
#     push!(qref,  [-0.4679730288403753, 0.3378517519243948,
 ↪  3.0742776436895087, -2.1493325227177693, -0.5149645531366437,
 ↪  1.1773161545883195, 1.8352431320081424])
#     push!(qref,  [-0.5052373634437477, 0.34403729863692983,
 ↪  3.06903925233866, -2.1397184251878762, -0.5559645531366437,
 ↪  1.1946313123156016, 1.856362359268435])
```

```
#     push!(qref,  [-0.54250169804712, 0.3502228453494649,
 ↪  3.063800860987811, -2.1301043276579827, -0.5969645531366438,
 ↪  1.2119464700428837, 1.8774815865287278])
#     push!(qref,  [-0.5797660326504925, 0.3564083920619999,
 ↪  3.0585624696369624, -2.1204902301280892, -0.6379645531366438,
 ↪  1.2292616277701658, 1.8986008137890207])
#     push!(qref,  [-0.6170303672538648, 0.36259393877453494,
 ↪  3.0533240782861135, -2.1108761325981957, -0.6789645531366438,
 ↪  1.2465767854974479, 1.9197200410493134])
#     push!(qref,  [-0.6542947018572371, 0.36877948548706996,
 ↪  3.0480856869352646, -2.1012620350683022, -0.7199645531366438,
 ↪  1.26389194322473, 1.940839268309606])
#     push!(qref,  [-0.6915590364606096, 0.37496503219960503,
 ↪  3.0428472955844157, -2.091647937538409, -0.7609645531366438,
 ↪  1.281207100952012, 1.961958495569899])
#     push!(qref,  [-0.728823371063982, 0.38115057891214005,
 ↪  3.0376089042335668, -2.0820338400085157, -0.8019645531366438,
 ↪  1.2985222586792942, 1.983077722830192])
#     push!(qref,  [-0.7660877056673544, 0.3873361256246751,
 ↪  3.0323705128827183, -2.072419742478622, -0.8429645531366439,
 ↪  1.3158374164065763, 2.0041969500904844])
#     push!(qref,  [-0.8033520402707267, 0.39352167233721014,
 ↪  3.0271321215318694, -2.0628056449487286, -0.8839645531366439,
 ↪  1.3331525741338583, 2.0253161773507773])
#     push!(qref,  [-0.8406163748740991, 0.39970721904974515,
 ↪  3.0218937301810205, -2.053191547418835, -0.9249645531366439,
 ↪  1.3504677318611407, 2.04643540461107])
#     push!(qref,  [-0.8778807094774715, 0.4058927657622802,
 ↪  3.0166553388301716, -2.043577449888942, -0.965964553136644,
 ↪  1.3677828895884225, 2.067554631871363])
#     push!(qref,  [-0.915145044080844, 0.41207831247481524,
 ↪  3.0114169474793226, -2.0339633523590486, -1.006964553136644,
 ↪  1.3850980473157049, 2.0886738591316556])
#     push!(qref,  [-0.9524093786842164, 0.4182638591873503,
 ↪  3.006178556128474, -2.024349254829155, -1.0479645531366442,
 ↪  1.402413205042987, 2.1097930863919485])
#     push!(qref,  [-0.9896737132875887, 0.42444940589988533,
 ↪  3.0009401647776253, -2.0147351572992616, -1.088964553136644,
 ↪  1.419728362770269, 2.1309123136522414])
#     push!(qref,  [-1.026938047890961, 0.43063495261242035,
 ↪  2.9957017734267763, -2.005121059769368, -1.1299645531366442,
 ↪  1.4370435204975511, 2.1520315409125343])
#     push!(qref,  [-1.0642023824943334, 0.43682049932495537,
 ↪  2.9904633820759274, -1.9955069622394748, -1.1709645531366442,
 ↪  1.4543586782248332, 2.173150768172827])
```

```
#     push!(qref,  [-1.101466717097706, 0.44300604603749044,
 ↪  2.9852249907250785, -1.9858928647095815, -1.2119645531366443,
 ↪  1.4716738359521153, 2.1942699954331197])
#     push!(qref,  [-1.1387310517010785, 0.4491915927500255,
 ↪  2.97998659937423, -1.976278767179688, -1.2529645531366442,
 ↪  1.4889889936793974, 2.215389222693412])
#     push!(qref,  [-1.1759953863044506, 0.45537713946256053,
 ↪  2.974748208023381, -1.9666646696497945, -1.2939645531366444,
 ↪  1.5063041514066795, 2.236508449953705])
#     push!(qref,  [-1.213259720907823, 0.46156268617509555,
 ↪  2.969509816672532, -1.9570505721199012, -1.3349645531366443,
 ↪  1.5236193091339616, 2.257627677213998])
#     push!(qref,  [-1.2478021278200226, 0.46729641720217435,
 ↪  2.9646540572035307, -1.9481387245716801, -1.3729697583408722,
 ↪  1.5396697023673502, 2.277204276379416])
#     push!(qref,  [-1.2748022884966306, 0.4717782027187738,
 ↪  2.9608585403999896, -1.9411727561509207, -1.402676624155692,
 ↪  1.5522155332738579, 2.292506375168055])
#     push!(qref,  [-1.2941678050105965, 0.47499270549426853,
 ↪  2.9581362549951593, -1.9361765052728712, -1.423983489970512,
 ↪  1.5612138684508734, 2.3034816078838536])
#     push!(qref,  [-1.3058986773619201, 0.4769399255286585,
 ↪  2.9564872009890393, -1.9331499719375316, -1.4368903557853316,
 ↪  1.566664707898397, 2.310129974526812])
#     push!(qref,  [-1.3099949055506017, 0.47761986282194374,
 ↪  2.95591137838163, -1.932093156144902, -1.4413972216001514,
 ↪  1.5685680516164289, 2.3124514750969296])
#     push!(qref,  [-1.3100000000000003, 0.47762070845504206,
 ↪  2.955910662235214, -1.9320918417907373, -1.4414028267565981,
 ↪  1.568570418791257, 2.312454362330413])

#     qref = typeof(zeros(7))[]
#     push!(qref,  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])
#     push!(qref,  [-0.07634644162642143, 0.012672827411047383,
 ↪  -0.010732313987104905, 0.019697175427098694, -0.084,
 ↪  0.03547495729491942, 0.04326866072840471])
#     push!(qref,  [-0.15269288325284286, 0.025345654822094766,
 ↪  -0.02146462797420981, 0.03939435085419739, -0.168, 0.07094991458983883,
 ↪  0.08653732145680942])
#     push!(qref,  [-0.22903932487926432, 0.03801848223314215,
 ↪  -0.032196941961314715, 0.05909152628129609, -0.252, 0.10642487188475827,
 ↪  0.12980598218521414])
#     push!(qref,  [-0.3053857665056857, 0.05069130964418953,
 ↪  -0.04292925594841962, 0.07878870170839478, -0.336, 0.14189982917967767,
 ↪  0.17307464291361885])
```

```
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
```

```
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#    push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
```

```
#     push!(qref,  [-0.3726433460337236, 0.061855467125350325,
 ↪  -0.05238391350848822, 0.09614097529893409, -0.41, 0.17315157727282096,
 ↪  0.21119227260292772])
#     push!(qref,  [-0.3081748275792907, 0.05115426887151827,
 ↪  -0.04332132502896183, 0.07950827192114293, -0.3390686581481977,
 ↪  0.14319578771253505, 0.17465531825059288])
#     push!(qref,  [-0.2318283859528692, 0.038481441460470875,
 ↪  -0.03258901104185691, 0.05981109649404422, -0.2550686581481976,
 ↪  0.1077208304176156, 0.13138665752218814])
#     push!(qref,  [-0.1554819443264477, 0.02580861404942348,
 ↪  -0.021856697054752, 0.04011392106694551, -0.17106865814819755,
 ↪  0.07224587312269615, 0.08811799679378339])
#     push!(qref,  [-0.07913550270002623, 0.013135786638376092,
 ↪  -0.011124383067647084, 0.0204167456398468, -0.08706865814819748,
 ↪  0.036770915827776696, 0.04484933606537864])
#     push!(qref,  [-0.002789061073605048, 0.00046295922732875194,
 ↪  -0.00039206908054222056, 0.0007195702127481829, -0.0030686581481977893,
 ↪  0.0012959585328574108, 0.0015806753369740911])
#     push!(qref,  [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0])

#     Xref = typeof(zeros(14))[]
#     for k = 1:length(qref)
#         push!(Xref, [qref[k]; qref[k]])
#     end
#     for k = 1:params.H
#         push!(Xref, [qref[end]; qref[end]])
#     end
#     return Xref
# end

# function callback(msg::JointTrajectoryPoint, x0, solve_mpc, params)
#     x0_new = zeros(params.n)
#     x0_new[1:7] .= msg.positions
#     x0_new[8:14] .= msg.velocities
#     if norm(x0_new - x0) >= 1e-6
#         x0 .= x0_new
#         solve_mpc .= true
#     end
# end

function traj_callback(msg::arthur_traj, Xref, x0)
    Xref[1] = typeof(zeros(14))[]
    for k=1:length(msg.traj.points)
        q = msg.traj.points[k].positions
        q = msg.traj.points[k].velocities
        push!(Xref[1], [q; q])
    end
    # if norm(x0 - zeros(14)) == 0
    #     copy!(x0, Xref[1][1])
    # end
    # RobotOS.loginfo("In Callback")
    # println(Xref[1][1])
end

function callback_x0(msg::JointState, x0, solve_mpc)
    x0_new = zeros(14)
    x0_new[1:7] .= msg.position
    x0_new[8:14] .= msg.velocity
    if norm(x0_new - x0) >= 1e-6
        x0 .= x0_new
        solve_mpc .= true
    end
end

function loop(sub_obj1, sub_obj2, pub_obj, x0, Xref, params, solve_mpc)
    loop_rate = Rate(1.0/params.dt)
    prob_mpc = ArthurHorizonProblem(Xref[1], x0, params.H, start=1)
    altro_mpc = ALTROSolver(prob_mpc, params.opts)
    solve!(altro_mpc)
    while ! is_shutdown()
        RobotOS._run_callbacks(sub_obj1)
        RobotOS._run_callbacks(sub_obj2)
        JointTrajectoryOutput = JointTrajectory()
        if norm(x0 - Xref[1][end]) > 0.05 && solve_mpc[1]
            # t0 = RobotOS.to_sec(RobotOS.now())
            println(x0)
            # mpc_update(altro_mpc, prob_mpc, Z_track, x0, t0)
            k_mpc = argmin(norm.([(Xref[1][k][1:7] - x0[1:7]) for
             ↪  k=1:length(Xref[1])]))
            # k_mpc = 1
            println(k_mpc)
            prob_mpc = ArthurHorizonProblem(Xref[1], x0, params.H,
             ↪  start=k_mpc)
            altro_mpc = ALTROSolver(prob_mpc, params.opts)
            solve!(altro_mpc)
            X = states(altro_mpc)
            U = controls(altro_mpc)
            points = Array{JointTrajectoryPoint}(undef, length(U))
            for k = 1:length(U)
                point = JointTrajectoryPoint()
                point.positions = X[k][1:7]
                point.velocities = X[k][8:14]
                point.effort = U[k]
                points[k] = point
            end
            JointTrajectoryOutput.header.stamp = RobotOS.now()
            JointTrajectoryOutput.points = points
            # publish(pub_obj, JointTrajectoryOutput)
            # solve_mpc .= false
        end
        publish(pub_obj, JointTrajectoryOutput)
        rossleep(loop_rate)
    end
end

# function loop(sub_obj1, sub_obj2, params, x0, Xref)
#     loop_rate = Rate(1.0/params.dt)
#     while ! is_shutdown()
#         RobotOS._run_callbacks(sub_obj1)
#         RobotOS._run_callbacks(sub_obj2)
#         println(x0)
#         println(Xref[1][1])
#         rossleep(loop_rate)
#     end
# end

function main()
    init_node("MPC_Node")
    params = MPC_Params()
    Xref = [typeof(zeros(14))[]]
    # Xref = get_trajectory(params)
    # x0 = copy(Xref[1])
    solve_mpc = [true]
    x0 = zeros(14)
    pub = Publisher{JointTrajectory}("joint_torques",queue_size=1)

    sub_x0 = Subscriber{JointState}("/my_gen3/joint_states", callback_x0,
     ↪  (x0,solve_mpc), queue_size=1)
    sub_traj =
     ↪  Subscriber{arthur_traj}("/my_gen3/arthur_traj",traj_callback,(Xref,x0),queue_siz
    # sub = Subscriber{JointTrajectoryPoint}("state",callback,(x0,
     ↪  solve_mpc, params),queue_size=1)

    while length(Xref[1]) == 0
        RobotOS._run_callbacks(sub_traj)
    end
    while norm(x0 - zeros(14)) == 0
        RobotOS._run_callbacks(sub_x0)
    end

    # prob = ArthurProblem(Xref[1], params=params)
    # altro = ALTROSolver(prob, params.opts)
    # solve!(altro)
    # Z_track = TrajectoryOptimization.get_trajectory(altro)

    # t0 = RobotOS.to_sec(RobotOS.now())
    # TrajectoryOptimization.set_initial_time!(prob, t0)
    # TrajectoryOptimization.set_initial_time!(prob_mpc, t0)

    loop(sub_traj, sub_x0, pub, x0, Xref, params, solve_mpc)
    # loop(sub_traj, sub_x0, params, x0, Xref)
end

if !isinteractive()
    main()
end
```