Individual Lab Report 4 - Progress Review 3

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Anthony Kyu

Team C:

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

March 24th, 2022

Carnegie Mellon University
The Robotics Institute

Contents

1 Individual Progress

Since the last progress review (Progress Review 2), I was responsible for further developing the MPC and getting an initial MPC up and running for simulation.

During the first week of this two week sprint, I spent time on re-examining used libraries to implement previously overlooked functions, iterating and testing newly implemented code. However, because this code was complex, it was initially hard to debug. Therefore, I then swapped to simplifying the dynamics model to only include the joint position and velocity states, and optimizing that initial model referencing examples from the Optimal Control and Reinforcement Learning that Professor Manchester recommended.

The solver for the MPC (ALTRO) hinged a lot on the initial guess for the input/torque trajectory, so the second week was spent trying several approaches to creating that initial guess to allow the MPC to converge. The final and successful approach was to calculate the gravity compensation torques for each joint at each position in the trajectory, which allowed the MPC to converge in about 200 ms on warm-start and about 1.3 seconds on cold-start. Warm-start uses the output trajectory from the previous run of the ALTRO solver as an initial guess.

After achieving this performance, I worked on writing code to visualize the output of the MPC (Figure 1), and optimizing performance. Once a visualizer was created, trajectory error was calculated and was determined to be extremely low. Collaborating with Sundaram, we were able to determine the error at the end-effector as well. Table 1 shows the error of the end-effector position at the end of the trajectory while using the MPC compared to the desired trajectory. Similar results were also found throughout the trajectory.

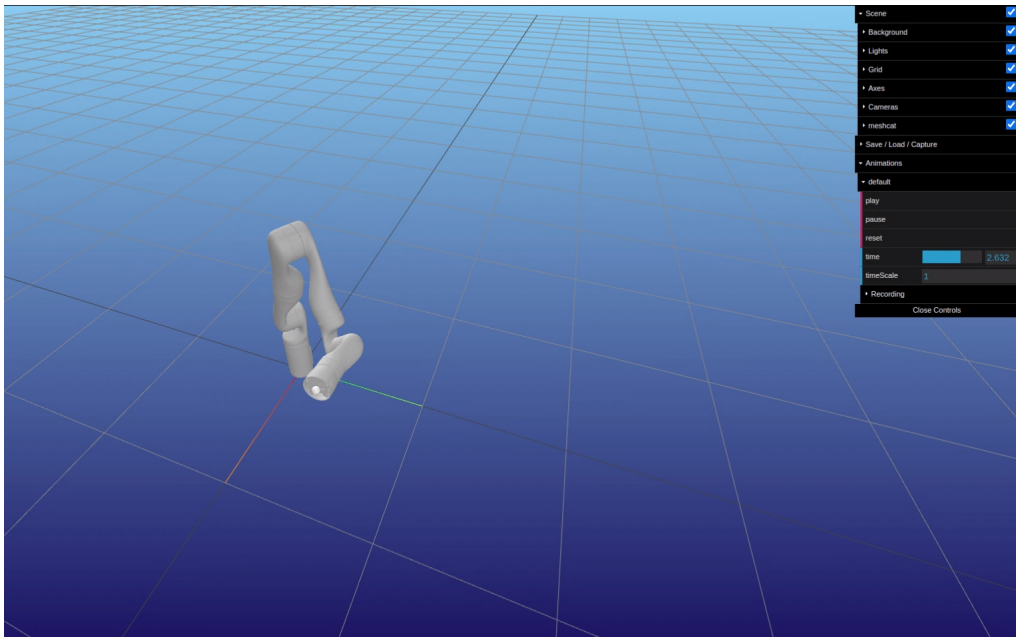

Figure 1: Simulation Visualizer of MPC Controlling Kinova Gen 3 Arm

Table 1: The end-effector pose error of at the end of the trajectory when comparing MPC output to desired trajectory.

Poses and Pose Error Relative to World Frame		
	Final Position (m)	**Final Orientation (RPY - degrees)**
Desired Trajectory	[0.214, 0.399, 0.436]	[89.764, -1.893, 91.168]
Model Predictive Controller	[0.214, 0.399, 0.436]	[89.764, -1.320, 91.166]
Norm of Error	0.000 mm	0.573 degrees
Joint State Error:	[0.00015959433561874015, -0.01370861054813316, -0.00020415288308361923, -0.006550863538767437, 0.004910735548274481, -0.0025631207105871745, -0.0039981932771855355, 1.1578955204986224e-5, 0.0025025376776104894, 5.748741903614427e-5, 4.2852715328442655e-5, -0.00028492385974444694, 2.2142163532343218e-6, 0.0001753074183847769]	

I also spent time testing the MPC's ability to optimize a trajectory given strict constraints (Figure 2). When the constraints on joint velocity were low (velocity constraint was lower than the desired velocity), the MPC did hold to those constraints as shown in Figure 3.

```
# State Bounds based on Robot Specs (Joint velocity and speed limits)
x_bnd = zeros(n)
x_bnd[1:7] = [Inf, deg2rad(128.9), Inf, deg2rad(147.8), Inf, deg2rad(120.3), Inf] # rad
x_bnd[8:14] = [0.25, 0.05, 0.05, 0.05, 0.25, 0.15, 0.15] # rad/sec
state_bnd = BoundConstraint(n,m, x_min=-x_bnd, x_max=x_bnd)
add_constraint!(conSet, state_bnd, 1:N)
```

Figure 2: Strict Constraints on the MPC - Velocity Constraints are very limiting.

```
x0:     [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0]
x1:     [-0.25005, 0.05, -0.01045, 0.05002, -0.25007, 0.15, 0.15002]
x2:     [-0.25005, 0.05, -0.03277, 0.05002, -0.25007, 0.15, 0.15002]
x3:     [-0.25005, 0.05, -0.04515, 0.05002, -0.25007, 0.15001, 0.15002]
x4:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x5:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x6:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x7:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x8:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x9:     [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x10:    [-0.25005, 0.05, -0.05, 0.05002, -0.25007, 0.15, 0.15002]
x11:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x12:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x13:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x14:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x15:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x16:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x17:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x18:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x19:    [-0.25005, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x20:    [-0.25004, 0.05, -0.05, 0.05002, -0.25006, 0.15, 0.15002]
x21:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x22:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x23:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x24:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x25:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x26:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x27:    [-0.25004, 0.05, -0.05, 0.05002, -0.25005, 0.15, 0.15002]
x28:    [-0.25003, 0.05, -0.05, 0.05002, -0.25004, 0.15, 0.15002]
x29:    [-0.25003, 0.05, -0.05, 0.05001, -0.25004, 0.15, 0.15002]
x30:    [-0.25003, 0.05, -0.05, 0.05001, -0.25004, 0.15, 0.15001]
x31:    [-0.25003, 0.05, -0.05, 0.05001, -0.25004, 0.15, 0.15001]
x32:    [-0.25003, 0.05, -0.05, 0.05001, -0.25004, 0.15, 0.15001]
x33:    [-0.25003, 0.05, -0.05, 0.05001, -0.25004, 0.15, 0.15001]
x34:    [-0.25003, 0.05, -0.05, 0.05001, -0.25003, 0.15, 0.15001]
x35:    [-0.25002, 0.05, -0.05, 0.05001, -0.25003, 0.15, 0.15001]
x36:    [-0.25002, 0.05, -0.05, 0.05001, -0.25003, 0.15, 0.15001]
x37:    [-0.25002, 0.05, -0.04045, 0.05001, -0.25003, 0.15, 0.15001]
x38:    [-0.25002, 0.04684, -0.02876, 0.05001, -0.25003, 0.1346, 0.15001]
x39:    [-0.25002, 0.03409, -0.017, 0.05001, -0.25002, 0.09305, 0.15001]
x40:    [-0.25002, 0.02254, -0.00564, 0.05001, -0.25002, 0.05207, 0.15001]
x41:    [-0.01554, 0.00194, -0.0003, 0.00709, -0.02177, 0.00197, 0.00764]
```

Figure 3: The joint velocities of the MPC resulting from the strict constraints.

Lastly, I started implementing code to connect the MPC to ROS using RobotOS.jl library within Julia, which is still a work in progress. All code for this progress review is in Appendix A and B.

2 Challenges

The major challenges for this progress review were getting the MPC to run without runtime errors, and then to get the MPC to converge such that the optimal torque trajectory did not cause the system to become unstable.

To meet the first challenge, I re-examined previously overlooked library functions and simplified the dynamics model to decrease likelihood of bugs in the dynamics function. To meet the second challenge, the initial torque/input trajectory guess was extremely important to determining convergence. Several approaches to tested, which included initializing the inputs to zero, initializing the inputs to match the interpolated accelerations of the trajectory, and finally initializing the inputs to match the torques required for gravity compensation at each position in the trajectory. The final approach (gravity compensation) was the successful one, and allowed the MPC to converge to a desired solution.

The last challenge was that I had overestimated the amount of work to meet the test requirements of testing the MPC in simulation. In order to mitigate this challenge, I informed my team of the challenges ahead of time so that they could help where they could as well as set realistic goals for myself.

3 Team Work

3.1 Anthony Kyu

Anthony worked on further developing the Model Predictive Controller, refactoring the code to use previously overlooked functions implemented in the RigidBodyDynamics.jl Library, and simplifying the dynamics model to only include joint positions and velocities in the state to simplify the overall MPC to get an initial code base. Then, in order to get the MPC to converge, Anthony worked to create an appropriate initial guess for the input/torque trajectory based on the given state trajectory, working on several approaches. Once the MPC converged Anthony worked on visualization, optimization and initial code implementation of the MPC into ROS using RobotOS.jl library in Julia. Anthony also collaborated with Sundaram to get a sample trajectory to work with and sanity checked Parker's PCB.

3.2 Parker Hill

Parker worked on creating a end-effector reaming adapter for this progress review, going through the process of designing, 3D printing, assembling, and analyzing the assembly. He also worked on the PCB assignment, going through the steps to create the board, find parts, and verify performance for submission. Finally, Parker helped to perform the motor speed and torque test as well as helped to create a ROS node which runs using Julia.

3.3 Sundaram Seivur

Sundaram worked on generating the trajectory for the arm to follow and worked on setting up the necessary packages and functions to do so. He first worked on setting up the arm and connecting it with a computer via ethernet. To do this, he set up the IP address of the system and validated if the joint parameters are visible via the KINOVA API. Sundaram also worked on setting up the Pilz Industrial Motion Planner in MoveIt! to be used as a trajectory planner. This was a involved process as a completely new planning-pipeline had to be set up and had to be included in the convoluted code written by Kinova. He also changed the IK solver from KDL to IKFast for which he set up a docker image and created an IKFast plugin. With all the improvements, Sundaram was successfully able to generate deterministic trajectories and send these commands to the arm to validate it's motion. Sundaram also collaborated with Anthony to send joint-state trajectories to the MPC controller and computed the cartesian pose of the end-effector.

3.4 Kaushik Balasundar

Kaushik worked alongside Gunjan in developing the landmark capture capability to convert the marker pose into pointcloud2 messages. This was then visualized on RViz and verified with ground truth to ensure the data is of the right scale. He also modified the URDF to incorporate new design changes and also added a force-torque plugin into the Gazebo simulation. He assisted Sundaram in extracting a trajectory planned by the Pilz planner for preliminary testing of the MPC controller.

3.5 Gunjan Sethi

Gunjan worked on developing the landmark capture functionality along with Kaushik. During this, the main tasks involved understanding the PointCloud2 message type in ROS, prototyping a simple pointcloud collection function and visualizing it in RViz. Later, they performed a test to verify the scale of the collected pointcloud and to study the effect of orientation of registration probe on the collected points. Gunjan also prepared slides for and presented the project management portion of the PDR.

4 Plans

For the next progress review, I plan to further optimize the MPC, and get a basic MPC node running in ROS to get initial performance metrics. I also plan to collaborate with Sundaram to create a testing pipeline to measure the trajectory errors and MPC performance. This pipeline would also include a full integration with Gazebo and ROS. And lastly, I would like to re-implement states into the MPC such as cartesian end-effector pose and velocities, and the applied forces, and be able to test the updated MPC in Gazebo simulation, specifically testing for the force constraints.

5 Appendix A: Model and Dynamics for Arthur Robot (Kinova Gen 3 - 7 DoF Arm)

Please note that the following code is currently being developed, and is not complete.

```
using RigidBodyDynamics
using StaticArrays
using Parameters
using RobotDynamics
using Rotations
using LinearAlgebra
using ForwardDiff

# Defining Arthur model using RigidBodyDynamics
struct Arthur{C} <: AbstractModel
    mechanism::Mechanism{Float64}
    statecache::C
    dyncache::DynamicsResultCache{Float64}
    function Arthur(mechanism::Mechanism)
        statecache = StateCache(mechanism)
        rescache = DynamicsResultCache(mechanism)
        new{typeof(statecache)}(mechanism, statecache, rescache)
    end
end

#TODO: Change Path
Arthur(;
 ↪  mechanism=RigidBodyDynamics.URDF.parse_urdf("../../../kortex_description/arms/gen3/7
 ↪  copy.urdf", remove_fixed_tree_joints = false)) = Arthur(mechanism)

# State, s, is [q q x x F]
# x will be input from the camera
# q, q, x will be taken or derived from the arm
# F will be input from the F/T Sensor
# Input, u, is Torque ()
function RobotDynamics.dynamics(model::Arthur, x::AbstractVector{T1},
 ↪  u::AbstractVector{T2}) where {T1,T2}
    # Create a state of the mechanism model and a result struct for the
     ↪  dynamics
    T = promote_type(T1,T2)
    state = model.statecache[T]
    res = model.dyncache[T]

    # Get states and constants of system not dependent on model state
    # num_q = RigidBodyDynamics.num_positions(model.mechanism)
```

```
    # q = x[SA[1, 2, 3, 4, 5, 6, 7]]
    # q = x[SA[8, 9, 10, 11, 12, 13, 14]]
    # p = x[2*num_q + 1:2*num_q + 6]
    # p = x[2*num_q + 7:2*num_q + 12]
    # F = x[2*num_q + 13:2*num_q + 18]
    # Be = zeros(T, 6, 6)

    # if (norm(p) > 1e-5)
    #     for k = 1:6
    #         Be[k,k] = norm(F) / norm(p)
    #     end
    # end

    # Set mechanism state to current state
    copyto!(state, x[SA[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14]])

    # w = Wrench{T}(default_frame(bodies(model.mechanism)[end-1]), F[4:6],
     ↪ F[1:3])
    # wrenches = BodyDict{Wrench{T}}(b => zero(Wrench{T},
     ↪ root_frame(model.mechanism)) for b in bodies(model.mechanism))
    # wrenches[bodies(model.mechanism)[end-1].id] = transform(w,
     ↪ transform_to_root(mechanismState, bodies(model.mechanism)[end-1]))
     ↪ dynamics!(dynamicsResult, mechanismState, u, wrenches)

    dynamics!(res, state, u)

    # p = [res.accelerations[bodies(model.mechanism)[end].id].linear;
     ↪ res.accelerations[bodies(model.mechanism)[end].id].angular]
    # F = Be*p
    # return SVector{32}([q; q; p; p; F])
    return SVector{14}([x[SA[8, 9, 10, 11, 12, 13, 14]]; res.v])
end

RobotDynamics.state_dim(::Arthur) = 14
RobotDynamics.control_dim(::Arthur) = 7
```

6 Appendix B: Model Predictive Controller Implementation (Still Work in Progress)

Please note that the following code is currently being developed, and is not complete.

```
import Pkg; Pkg.status()
include("../src/Arthur.jl")
using RigidBodyDynamics
using StaticArrays
using RobotDynamics
using Altro
using TrajectoryOptimization

model = Arthur()
n,m = size(model)

tf = 4.1
dt = 0.1
N = Int(round(tf/dt) + 1)

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,
 ↪  -1.995506962394748, -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,
 ↪  -0.0003920690805422056, 0.0007195702127481829, -0.003068658148197893,
 ↪  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:N
    push!(Xref, [qref[k]; qref[k]])
end

using MeshCat, MeshCatMechanisms, Blink
urdf = "../../../kortex_description/arms/gen3/7dof/urdf/GEN3_URDF_V12
 ↪  copy.urdf"
body = findbody(model.mechanism, "Bracelet_Link")
point = Point3D(default_frame(body), 0., 0, -0.07)
# Create the visualizer
traj_vis = MechanismVisualizer(model.mechanism, URDFVisuals(urdf))

# Render our target point attached to the robot as a sphere with radius 0.07
setelement!(traj_vis, point, 0.01)

qs = Vector{Float64}[]
for x in Xref
    push!(qs, copy(x[1:7]))
end
ts = collect(0:dt:(N-1)*dt);

setanimation!(traj_vis, Animation(traj_vis, ts, qs))
render(traj_vis)

# TODO:
# Uref could be just U equilibrium (just gravity compensation),
# Xref could be the desired joint and cartesian positions and velocities,
 ↪  and then 0 for the forces

state = MechanismState(model.mechanism)
zero!(state)
v = similar(velocity(state))
for joint in joints(state.mechanism)
    if joint == joints(state.mechanism)[2] || joint ==
     ↪  joints(state.mechanism)[3] || joint == joints(state.mechanism)[4] ||
     ↪  joint == joints(state.mechanism)[5] || joint ==
     ↪  joints(state.mechanism)[6] || joint == joints(state.mechanism)[7] ||
     ↪  joint == joints(state.mechanism)[8]
        v[joint][1] = 0.0
    end
end
```

```
Qref = Vector{Float64}[]
Uref = Vector{Float64}[]
push!(Qref, copy(Xref[1][1:14]))
for k = 1:N-1
    set_configuration!(state, Qref[k][1:7])
    set_velocity!(state, Qref[k][8:end])
    push!(Uref, inverse_dynamics(state, v))
    push!(Qref, Qref[k] + dynamics(model, Qref[k], Uref[k])*dt)
end

dtref = [dt for k=1:N]
traj = RobotDynamics.Traj(Xref, Uref, dtref, cumsum(dtref) .- dtref[1]);

# Set up
# TODO: Tune weights
Q = 100.0*Diagonal(@SVector ones(n))
Qf = 100*Diagonal(@SVector ones(n))
R = 1.0e-1*Diagonal(@SVector ones(m))
obj = TrajectoryOptimization.TrackingObjective(Q, R, traj, Qf=Qf)

# Create Empty ConstraintList
conSet = ConstraintList(n,m,N)

# Control Bounds based on Robot Specs (Joint torque limits)
u_bnd = [39.0, 39.0, 39.0, 39.0, 9.0, 9.0, 9.0]
control_bnd = BoundConstraint(n,m, u_min=-u_bnd, u_max=u_bnd)
add_constraint!(conSet, control_bnd, 1:N-1)

# State Bounds based on Robot Specs (Joint velocity and speed limits)
x_bnd = zeros(n)
x_bnd[1:7] = [Inf, deg2rad(128.9), Inf, deg2rad(147.8), Inf, deg2rad(120.3),
 ↪  Inf] # rad
x_bnd[8:14] = [1.39, 1.39, 1.39, 1.39, 1.22, 1.22, 1.22] # rad/sec
# x_bnd[8:14] = [0.25, 0.05, 0.05, 0.05, 0.25, 0.15, 0.15] # rad/sec
# x_bnd[15:end] = [Inf for k=1:(n-14)] # Constraints on force elsewhere
state_bnd = BoundConstraint(n,m, x_min=-x_bnd, x_max=x_bnd)
add_constraint!(conSet, state_bnd, 1:N)

# # Cartesian Velocity Bound
# x_max = 0.0005 # m/s
# vel_bnd = NormConstraint(n, m, x_max, Inequality(), 21:23)
# add_constraint!(conSet, vel_bnd, 1:N)

# # Force Bound (Fx Fy Fz)
# F_max = 20 # Newtons
```

```
# F_bnd = NormConstraint(n, m, F_max, Inequality(), 27:29)
# add_constraint!(conSet, F_bnd, 1:N)

# Other possible constraints:
# Cartesian angular velocity bounds?
# External moment bounds?

# Goal Constraint - only if you want the final state to be the desired state
# goal = GoalConstraint(xf)
# add_constraint!(conSet, goal, N)

prob = Problem(model, obj, Xref[end], tf, x0=Xref[1], constraints=conSet,
 ↪  X0=Xref, U0=Uref)

opts = SolverOptions(
    cost_tolerance_intermediate=1e-2,
    penalty_scaling=10.,
    penalty_initial=1.0,
#     projected_newton=false,
)

altro = ALTROSolver(prob, opts)
set_options!(altro, show_summary=true)
solve!(altro);

# Get some info on the solve
max_violation(altro)
cost(altro)
iterations(altro)

# Extract the solution
X = states(altro)
U = controls(altro)

# Extract the solver statistics
stats = Altro.stats(altro)   # alternatively, solver.stats
stats.iterations             # equivalent to iterations(solver)
stats.iterations_outer       # (number of Augmented Lagrangian iterations)
stats.iterations_pn          # (number of projected newton iterations)
stats.cost[end]              # terminal cost
stats.c_max[end]             # terminal constraint satisfaction
stats.gradient[end]          # terminal gradient of the Lagrangian
dstats = Dict(stats)         # get the per-iteration stats as a dictionary
 ↪  (can be converted to DataFrame)
```

```
using Printf
for i in 1:N
    print("x")
    print(i-1)
    print(":\t")
    println(round.(X[i][8:14]; digits=5))
end
println()

for i in 1:N-1
    print("u")
    print(i)
    print(": ")
    println(U[i])
end
println()

for i in 1:N
    print("x")
    print(i-1)
    print(": ")
    println(norm(X[i] - Xref[i]))
end
println()

print("Terminal Error: ")
println(norm(X[end] - Xref[end]))
println(X[end] - Xref[end])

# for i in 1:N-1
#     print("u")
#     print(i)
#     print(": ")
#     println(U[i] - Uref[i])
# end

using MeshCat, MeshCatMechanisms, Blink
urdf = "../../../kortex_description/arms/gen3/7dof/urdf/GEN3_URDF_V12
 ↪  copy.urdf"
body = findbody(model.mechanism, "Bracelet_Link")
point = Point3D(default_frame(body), 0., 0, -0.07)
# Create the visualizer
vis = MechanismVisualizer(model.mechanism, URDFVisuals(urdf))

# Render our target point attached to the robot as a sphere with radius 0.07
```

```
setelement!(vis, point, 0.01)

qs = Vector{Float64}[]
for x in X
    push!(qs, copy(x[1:7]))
end
ts = collect(0:dt:(N-1)*dt);

setanimation!(vis, Animation(vis, ts, qs))
render(vis)
```