# Individual Lab Report 3 - Progress Review 2

## Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR


Anthony Kyu

Team C:

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


March 3rd, 2022


Carnegie Mellon University
The Robotics Institute

## Contents

# 1 Individual Progress

Since the last progress review (Progress Review 1), I was responsible for iterating and completing the high-level control architecture, the optimal control problem (which includes defining the objective/cost function, the dynamics of the system, and the constraints of the system), explore and dive into different libraries to help implement the Model Predictive Controller (MPC), implement the MPC in simulation, and perform Test 13 (Test MPC for Force and Position Requirements in Simulation) by the end of the progress review. Unfortunately, not all of these goals were met because of challenges later discussed in this Independent Lab Report.

During the first week of this two week sprint, I spent time revising the optimal control problem and receiving feedback from Professor Zachary Manchester (professor of Optimal Control and Reinforcement Learning course) multiple times. Based on his feedback, the following high-level controls block diagram was finalized (Figure 1). q and q̇ will be read from the joint encoders (derive from these), $F_{External}$ will be read from the Force/Torque Sensor on the wrist, and X will be read from the external Atracsys 300 camera. Furthermore, the following optimal control problem was also finalized (Figure 2).

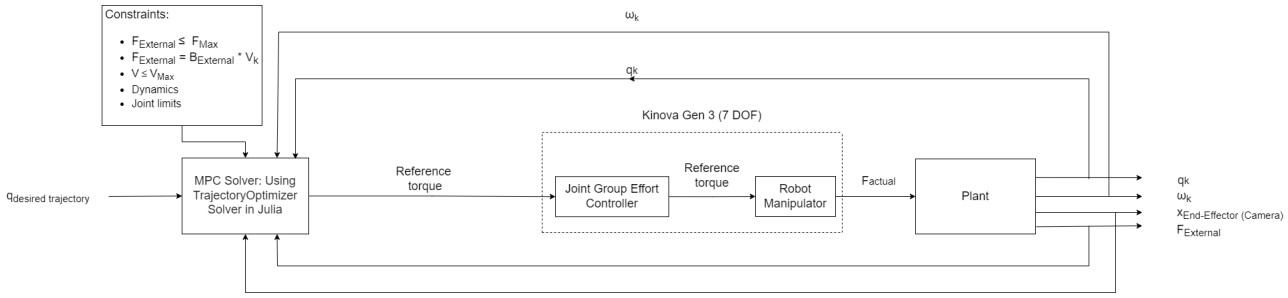


**Figure 1: High-Level Control Block Diagram**

$$\min_{s_k,u_k,k\epsilon[1,H]} \quad \frac{1}{2}(s_H - sd)^T Q_H (s_H - sd) + \sum_{k=1}^{H-1} \frac{1}{2}(s_k - sd)^T Q (s_k - sd) + \frac{1}{2} u_k^T R u_k$$

$$\text{s.t.} \quad \begin{bmatrix} \dot{q} \\ \ddot{q} \\ \dot{x} \\ \ddot{x} \\ \dot{F}_{External} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M^{-1}(\tau - \tau_{External} - C\dot{q} - G) \\ J\dot{q} \\ \dot{J}\dot{q} + J\ddot{q} \\ B_{External}(\dot{J}\dot{q} + J\ddot{q}) \end{bmatrix}$$

$$u_k = \tau \leq \tau_{Max}$$

$$||F_{External}|| = ||B_{External} J\dot{q}_k|| \leq F_{Max}$$

$$||\dot{X}|| = ||J\dot{q}_k|| \leq \dot{X}_{Max}$$

$$q_k \epsilon q_{Limits}$$

$$\dot{q}_k \epsilon \dot{q}_{Limits}$$

**Figure 2: The Optimal Control Problem in the MPC to Minimize**

After finalizing the optimal control problem, I started reading through the documentation of the Julia libraries I planned on using to implement the MPC, which includes RigidBodyDynamics, RobotDynamics, TrajectoryOptimization, and Altro. RigidBodyDynamics and RobotDynamics were used to implement the model and dynamics of the system, TrajectoryOptimization will be used set up the optimal control problem, and Altro will be used to solve the optimal control problem. While reading the documentation, I have also started implementing the model, dynamics, and constraints for our MPC (See Appendix A and B).

## 2 Challenges

The major challenge for this progress review was the uncertainty of not knowing what manipulator we were using for our project. From the previous progress review, we originally thought that we were going to use the UR5, but our sponsor decided that they wanted us to use a Kinova Gen 3 (7 DOF) instead. We did not learn about this until 6 days before the progress review, setting our plans back because we had to set up a new environment for simulation given the new arm, forcing us to push Test 13 (Test MPC for Force and Position Requirements in Simulation) back one progress review.

The other challenges during this progress review were from the scattered documentation of the libraries I was using to implement our code. This drastically slowed down development as well. In addition, some libraries had dependencies on older versions of other libraries, and making sure those matched was another challenge.

The last major challenge was making sure our optimal control problem was compatible with the libraries we were using. For example, the way the library wants us to implement constraints is to use variables directly from the state or controls. Originally, the state did not include $\dot{x}$, but we had to add it in and calculate the dynamics in order to implement the constraint on end-effector velocity. This challenge caused us to iterate on our optimal control problem even more, increasing development time.

# 3 Team Work

## 3.1 Anthony Kyu

Anthony worked on formulating the optimal control problem for the Model-Predictive Controller, creating several iterations of the optimal control problem and getting regular feedback from Professor Manchester. He also explored a variety of libraries to use for the MPC controller and for interfacing the controller (in Julia) with ROS. After that, he started implementing the MPC controller, coding the dynamics function, the constraints and the objective function. He also collaborated with Parker on the Power Distribution Design PCB, discussing requirements and ideas for the board, as well as researching some components for the board.

## 3.2 Parker Hill

Parker helped to set up the physical set-up for the Kinova Gen-3 arm which involved assembling a Vention table, picking up the arm from our sponsor, and setting up the arm on the table. He also designed and 3D printed prototypes for attaching the reamer handle to the end-effector. And lastly, he worked on the Power Distribution Board assignment, creating the conceptual design as well as the schematic of our motor control board.

## 3.3 Sundaram Seivur

Sundaram worked on formulating the optimal controls problem and collaborated with Anthony in getting feedback from professors. He studied the functions used to interface the output of the controls loop with ROS. He also worked on setting up the hardware which included getting the arm from our sponsors, assembling the Vention table and mounting the Gen3 arm. Finally, he worked with Kaushik to get a 3D of the model bone.

## 3.4 Kaushik Balasundar

Kaushik worked on implementing the iterative closest point registration algorithm and validating its efficacy in registering the points from the surface of the simulated acetabulum with the 3D scanned model of the pelvis. He and Sundaram 3D scanned the pelvis model using laser scanning equipment from Prof. Shimada's lab. Once the arm was finalized by our sponsors, he set up the simulation environment with the Kinova Gen-3 arm. He worked alongside Gunjan in publishing the marker poses to ROS and broadcasting the pose as a TF transform to visualize on RViz. Finally, he was our team's presenter for the second progress review.

## 3.5 Gunjan Sethi

Gunjan re-calibrated the markers to improve the robustness of the ROS camera node. Further, she added the marker pose detection and visualization features to the node and performed various reliability tests to ensure smooth functioning during the progress review.

## 4 Plans

For the next progress review, I plan to finish documenting the high-level design of the Optimal Control Problem, and the design rationale. I then plan to finish implementing the Model Predictive Controller, and set up the Gazebo environment to simulate the controller and test to see if it meets the force constraints and positional accuracy required. This would not only require some intial testing of the dyanmics model, but also integrating the MPC with ROS using RobotOS.jl library in Julia, and implementing some force modeling plugins into Gazebo to test the MPC.

As a stretch goal, I plan to start integrating the MPC in reality on our real Kinova Gen 3 Manipulator, working towards future progress reviews.

# 5 Appendix A: Currently Implemented Code for Model and Dynamics

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

# Defining Arthur model using RigidBodyDynamics
struct Arthur{T} <: AbstractModel
    mechanism::T
    function Arthur(mechanism)
        T = eltype(RigidBodyDynamics.Mechanism)
        new{T}(mechanism)
    end
end

Arthur(;
 ↪  mechanism=RigidBodyDynamics.URDF.parse_urdf("/home/amkyu/catkin_workspace/src
/ros_kortex/kortex_description/arms/gen3/7dof/urdf/GEN3_URDF_V12.urdf")) =
 ↪  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, s, u)
    # Create a state of the mechanism model and a result struct for the
     ↪  dynamics
    dynamicsResult = RigidBodyDynamics.DynamicsResult(model.mechanism)
    mechanismState = RigidBodyDynamics.MechanismState(model.mechanism)

    # Get states and constants of system not dependent on model state
    M = RigidBodyDynamics.mass_matrix(mechanismState)
    num_q = RigidBodyDynamics.num_positions(model.mechanism)
    q = s[1:num_q]
    q = s[num_q+1:2*num_q]
    x = s[2*num_q + 1:2*num_q + 6]
    x = s[2*num_q + 7:2*num_q + 12]
    F = s[2*num_q + 13:2*num_q + 18]
    Be = zeros(6, 6)
```

```
    if (norm(x) > 1e-5)
        for k = 1:3
            Be[k,k] = norm(F) / norm(x)
        end
    end

    # Set mechanism state to current state
    RigidBodyDynamics.set_configuration!(mechanismState, q)
    RigidBodyDynamics.set_velocity!(mechanismState, q)

    # Get variables dependent on state
    J = getJacobian(model, q, q)
    _ext = transpose(J)*Be*x

    # Calculate dynamics
    RigidBodyDynamics.dynamics!(dynamicsResult, mechanismState, u)
    # Add the effects of external forces/torques into dynamics
    q = M\((M * dynamicsResult.v) - _ext)
    x = getJ(model, J, q, q)*q + J*q
    F = Be*x
    return [q; q; x; F; 0; 0; 0; 0; 0; 0]
end

function getJacobian(model::Arthur, q,  q)
    mechanismState = RigidBodyDynamics.MechanismState(model.mechanism)
    RigidBodyDynamics.set_configuration!(mechanismState, q)
    RigidBodyDynamics.set_velocity!(mechanismState, q)
    p = RigidBodyDynamics.path(model.mechanism,
     ↪ RigidBodyDynamics.root_body(model.mechanism),
     ↪ RigidBodyDynamics.bodies(model.mechanism)[end])
    J_data = RigidBodyDynamics.geometric_jacobian(mechanismState, p)
    return [J_data.linear; J_data.angular]
end

function getX(model::Arthur, J, q, q)
    J = getJacobian(model, q, q)
    x = J*q
    return x
end

function getJ(model::Arthur, J, q, q)
   return ForwardDiff.jacobian(dq -> getX(model, J, dq, q), q)
end

RobotDynamics.state_dim(::Arthur) = 32
```

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

# 6 Appendix B: Currently Implemented Code for Objective Function and Constraints

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

```
include("Arthur.jl")
using RigidBodyDynamics
using StaticArrays
using RobotDynamics
using Rotations
using LinearAlgebra
using ForwardDiff, FiniteDiff
using Altro
using TrajectoryOptimization

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

N = 61
tf = 3.
dt = tf/(N-1)

x0 = @SVector zeros(n)
xf = @SVector zeros(n);  # i.e. swing up
#Traj = # Input from trajectory planner

# Set up
Q = 1.0e-2*Diagonal(@SVector ones(n))
Qf = 100.0*Diagonal(@SVector ones(n))
R = 1.0e-1*Diagonal(@SVector ones(m))
obj = TrackingObjective(Q, R, Traj, Qf=Qf)
# obj = LQRObjective(Q,R,Qf,xf,N);

# 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(26)
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[15:end] = [Inf, Inf, Inf, Inf, Inf, Inf, Inf, Inf, Inf, Inf, Inf, Inf]
 ↪  # 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(), 15:20)
add_constraint!(conSet, vel_bnd, 1:N)

# Force Bound
F_max = 20 # Newtons
F_bnd = NormConstraint(n, m, F_max, Inequality(), 21:26)
add_constraint!(conSet, F_bnd, 1:N)

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