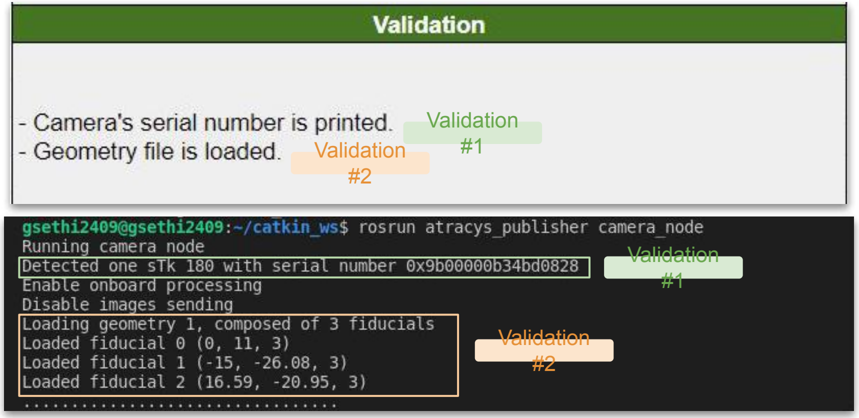
Autonomous Reaming for Total Hip Replacement (ARTHuR)

Progress Review - 2

Team C: Kaushik Balasundar, Parker Hill, Anthony Kyu, Sundaram Seivur, Gunjan Sethi

2 March, 2022

Previously



Validation of Test 1: Reading and displaying camera's serial number

$$\min_{u_k, k\in[1,H]} \frac{1}{2}(z_H - zd)^T Q (z_H - zd) + \sum_{k=1}^{H-1} \frac{1}{2}(z_k - zd)^T Q (z_k - zd) + \frac{1}{2} u_k^T R u_k$$

$$\text{s.t.} \quad \begin{bmatrix} x_{k+1} \\ \dot{x}_{k+1} \\ y_{k+1} \\ \dot{y}_{k+1} \\ z_{k+1} \\ \dot{z}_{k+1} \\ \phi_{k+1} \\ \dot{\phi}_{k+1} \\ \theta_{k+1} \\ \dot{\theta}_{k+1} \\ \psi_{k+1} \\ \dot{\psi}_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + \dot{x}_k h \\ \dot{x}_k + u_{x,k} h + F_{x,k} h \\ y_k + \dot{y}_k h \\ \dot{y}_k + u_{y,k} h + F_{y,k} h \\ z_k + \dot{z}_k h \\ \dot{z}_k + u_{z,k} h + F_{z,k} h \\ \phi_k + \dot{\phi}_k h \\ \dot{\phi}_k + u_{\phi,k} h + M_{\phi,k} h \\ \theta_k + \dot{\theta}_k h \\ \dot{\theta}_k + u_{\theta,k} h + M_{\theta,k} h \\ \psi_k + \dot{\psi}_k h \\ \dot{\psi}_k + u_{\psi,k} h + M_{\psi,k} h \end{bmatrix}$$

$$||F_{external}|| \le F_{Max}$$

$$||V_k|| \le V_{Max}$$

$$||x_k - x_d|| \le \varepsilon$$

$$||y_k - y_d|| \le \varepsilon$$

Preliminary controls formulation

Schedule

Schedule			
Identifier	**Capability Milestone(s)**	**Associated Tests**	**System Requirements**
✔ **Progress Review 1 2/16/2022**	- Test camera health and camera discovery via a ROS Node.	Test 1	M.F.1
Progress Review 2 3/2/2022	- Broadcast marker pose as a ROS transform & ROS topic - Validate the preliminary performance of the registration algorithm chosen	Test 2 Test 3 Test 4	M.F.1 M.F.3 M.N.1

Hardware: Robot Manipulator!

Figure: Workspace setup

Kinova Gen3

- Available until December 2022
- On-going discussions on porting code to Kinova Link-6 when APIs are made available

Progress Review #2 Tests

- ✓ Marker Pose Detection Test — Perception and Sensing
- ✓ Marker Pose Visualization Test — Perception and Sensing
- ✓ Preliminary Point Cloud Registration Test — Perception and Sensing
- ✓ Documentation — Perception and Sensing

Further Updates

- ✓ Registration — Perception and Sensing
- ✓ Controls — Planning and Controls
- ✓ Simulation — Planning and Controls
- ✓ Hardware — Hardware

Progress Review - 2
Tests

Test 2: Marker Pose Detection Test

Goal: Read camera measurements from a ROS Node, identify the fiducial points with a pre-loaded geometry file and print the 6DOF marker pose.

Approach: Add functionality onto the previously developed camera_node to detect marker poses using the provided Atracsys SDK.

Test 2:	
Marker Pose Detection Test	
Objective	
Test fiducial marker detection via a ROS Node.	
Equipment	MRSD Desktop 2, Atracys SpryTrack 300 Camera, Markers
Elements	Perception Subsystem
Personnel	Gunjan Sethi
Location	NSH Basement
Procedure	
1. Run the camera_node ROS Node and wait for the node to discover the camera. 2. Wait for the ROS Node to load the geometry file. 3. Place markers in front of the camera.	
Validation	
- Camera's serial number is printed. - Geometry file is loaded. - The marker pose is printed.	

Test 2: Marker Pose Detection Test

SNo.	Approach	Pros	Cons
1	Develop custom functions for triangulation and marker pose detection.	- Deep understanding of codebase. - Lighter implementation	- Complex s/w engineering issues - Writing low-latency code is difficult
2	Use marker pose detection APIs from Atracsys SDK	- Preprocessing of images not required. - Robust, well-tested codebase.	- Cannot resolve any latency blocks. - Less documentation - customization will be time-consuming

Test 2: Marker Pose Detection Test

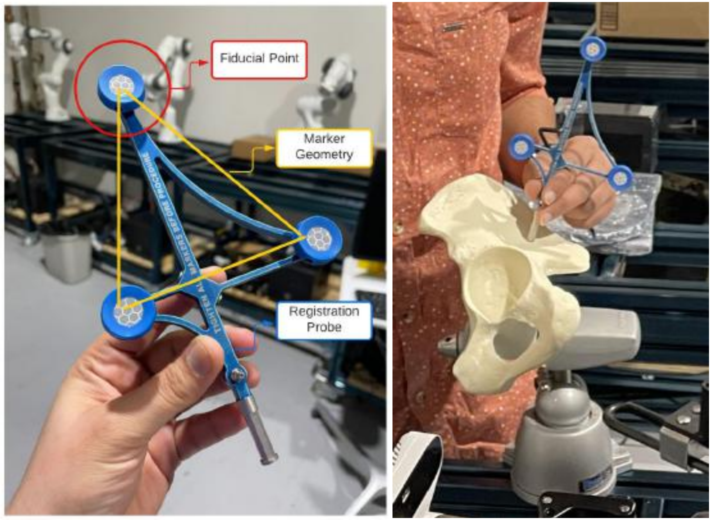


Figure: Marker (left) Usage of Marker on Registration Probe (right)

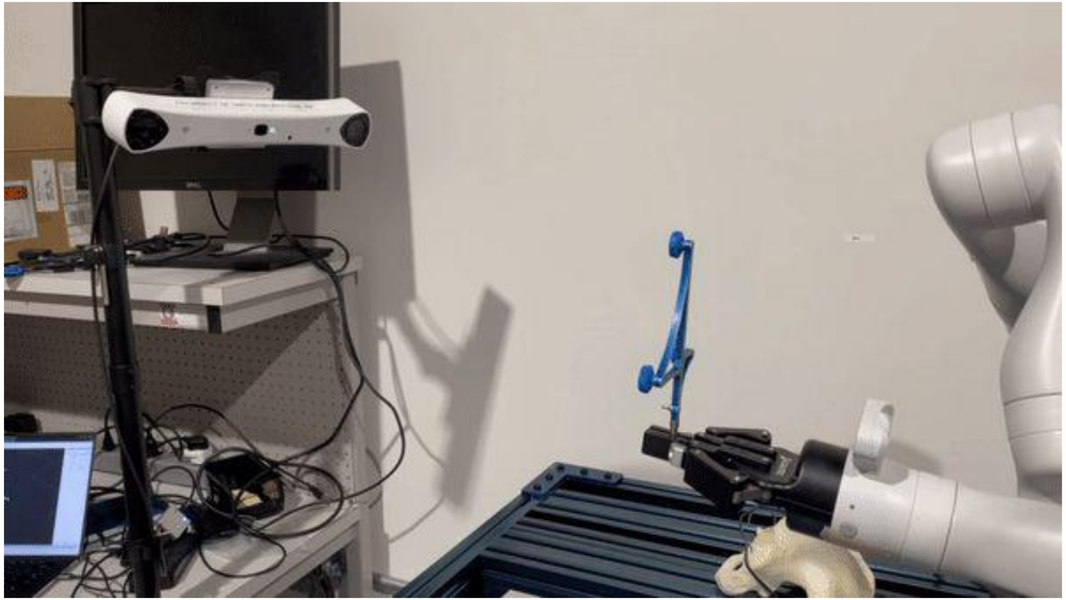


Figure: Test 2 Setup

Test 2: Marker Pose Detection Test

Results:

✓ Camera's serial number printed

✓ Geometry file loaded

✓ Marker pose printed in terminal

```
PROBLEMS 8   OUTPUT   TERMINAL   DEBUG CONSOLE

Right Count: 3
Fiducials Count- 0
Found!

geometry 9990
, rotation (-0.41 -0.91 0.05 -0.90 0.42 0.11 -0.12 -0.01 -0.99)
, trans (157.47 105.72 677.90), error 0.185

Publishing--
Raw Data Count-
Left Count: 3
Right Count: 3
Fiducials Count- 0
Found!

geometry 9990
, rotation (-0.41 -0.91 0.06 -0.90 0.42 0.11 -0.12 -0.01 -0.99)
, trans (157.42 104.99 678.79), error 0.185

Publishing--
Raw Data Count-
Left Count: 3
Right Count: 3
Fiducials Count- 0
Found!

geometry 9990
, rotation (-0.41 -0.91 0.06 -0.90 0.42 0.11 -0.12 -0.00 -0.99)
, trans (157.38 104.41 679.62), error 0.185
```

Test 2: Marker Pose Detection Test

Challenges:

➔ *The code was not reliable; marker would not be detected robustly.*

- Issue was resolved by marker recalibration
- Re-calibration of the marker geometry was performed using the GUI
- New geometry loaded onto the GUI to test the marker detection robustness
- The results were as expected within the error tolerance values

Test 2: Marker Pose Detection Test

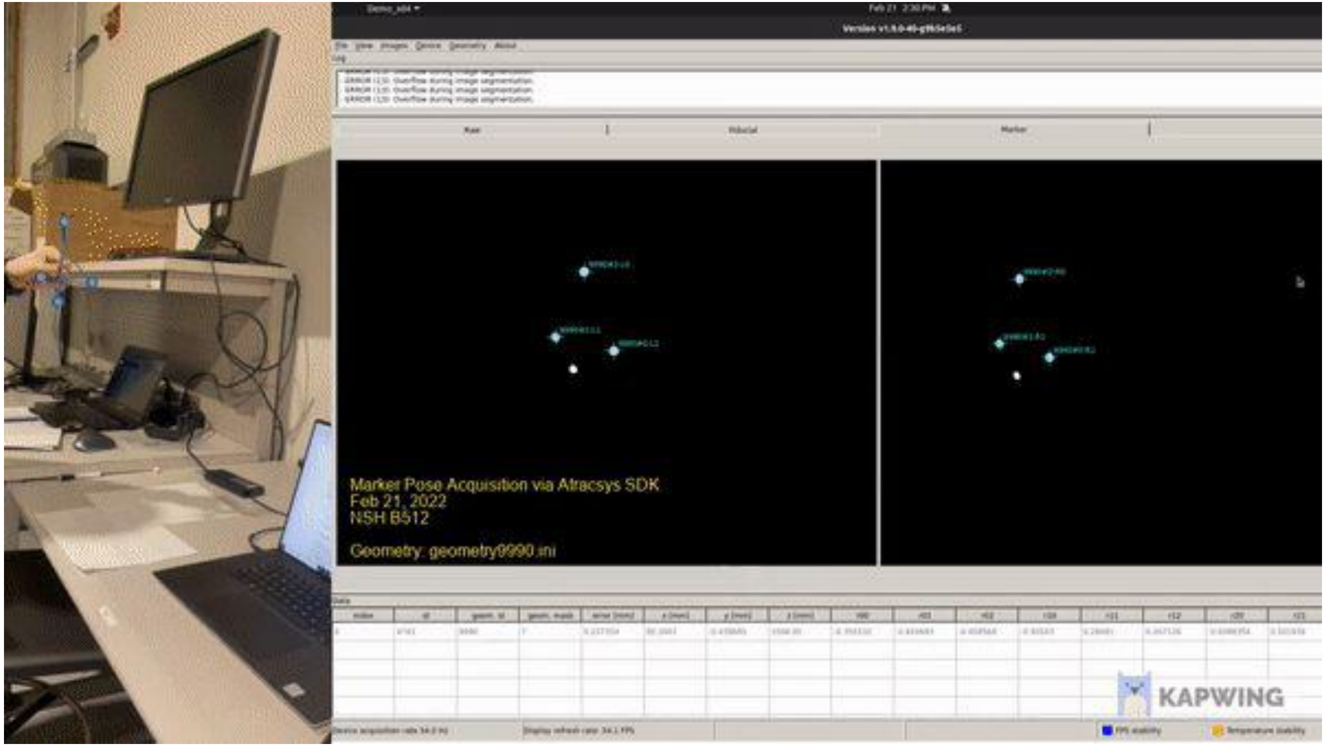


Figure: Marker Re-calibration Results

Test 3: Marker Pose Visualization Test

Goal:

- **Part 1.** Publish 6-DoF marker pose on a ROS topic & broadcast the transform.
- **Part 2.** Visualize the marker frame on RViz at >50Hz

Test 3:	
Marker Pose Visualization Test	
Objective	
Test the publishing of marker poses onto a ROS topic and visualizing markers using RViz.	
Equipment	MRSD Desktop 2, Atracys SpryTrack 300 Camera, Markers
Elements	Perception Subsystem
Personnel	Gunjan Sethi
Location	NSH Basement
Procedure	
1. Run the camera_node ROS Node and wait for the node to discover the camera. 2. Wait for the ROS Node to load the geometry file. 3. Place markers in front of the camera. 4. Run rostopic command-line tool to view messages on the marker-pose topic in ROS. 5. Run RViz	
Validation	
- Camera's serial number is printed. - Geometry file is loaded. - rostopic command line shows marker poses. - Markers appear on RViz.	

Test 3: Marker Pose Visualization Test

Approaches (Rotation Matrix to Quaternion)

Convert the incoming Marker frame to geometry_msgs/PoseStamped Message type.

SNo.	Approach	Pros	Cons
1	Use Eigen	- Well tested code - Low latency - Popular choice	- Typecasting to Eigen message to ROS message required
2	Use tf libraries (rot_to_quat)	- Well tested code - No typecasting required - Popular choice	- Deprecated functions; several dependency issues
3	Write a custom function	- Very simple implementation	- Need to perform robust testing and ensure FPS retention

Test 3: Marker Pose Visualization Test

Results

- ✓ Camera's serial number is printed.
- ✓ Geometry file is loaded.
- ✓ Pose published to a topic & appears on command-line (Part 1)
- ✓ Marker frame visualized on RViz (Part 2)

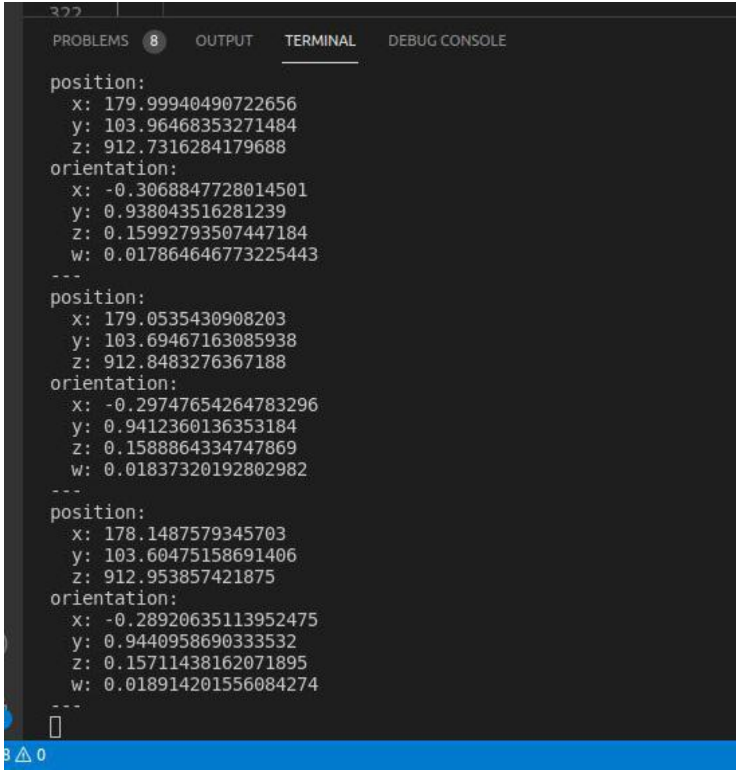
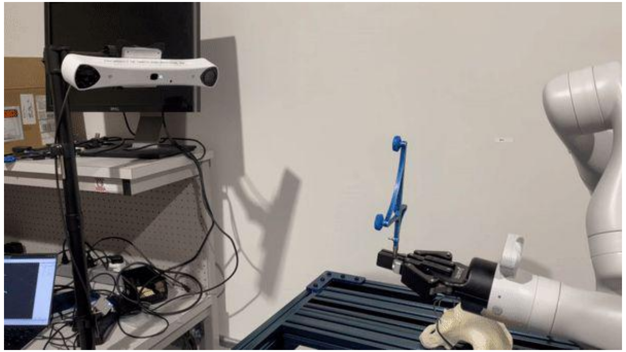
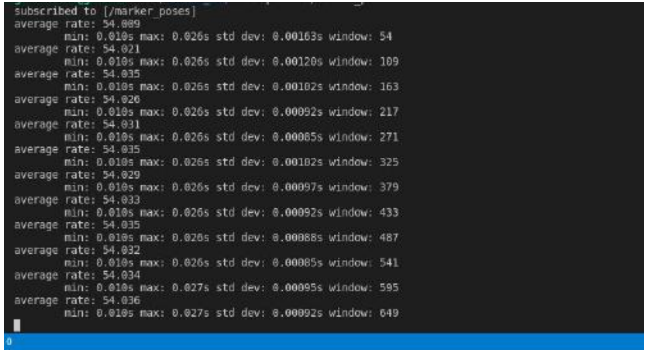


Figure: Marker Detection Test Results

Test 3: Marker Pose Visualization Test





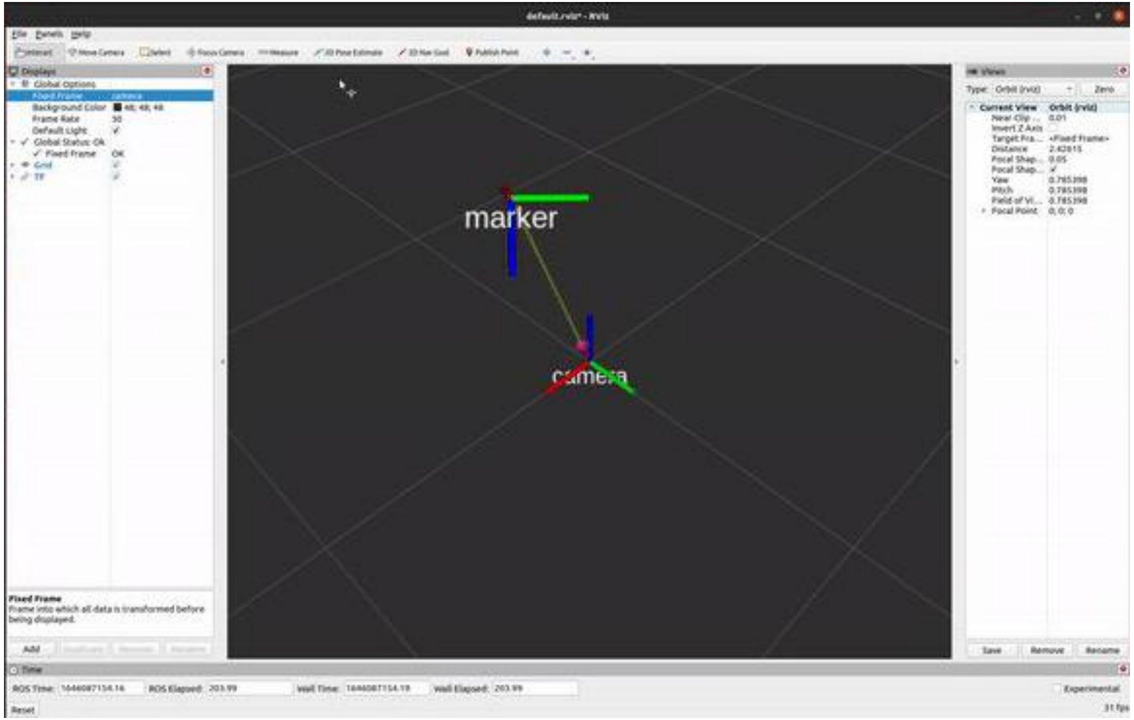


Figure: Marker TF Broadcasting on RViz

Test 3: Marker Pose Visualization Test

Challenges:

- ➔ TF functions not supported on TF2
  - ◆ Lack of functionality to convert rotation matrices to quaternion in TF2
  - ◆ TF2 required quaternion for rotation - did not support rotation matrices
- ➔ Eigen to ROS TF message conversion
  - ◆ Eigen outputs needed to be extracted and type-casted to ROS compatible dependencies
- ➔ Debugging Missing Dependencies
  - ◆ CMake Errors were always not indicative of the root issue - debugging this took time
- ➔ Marker TF not visible on RViz
  - ◆ Units conversions and flipping of coordinate axes

Test 4: Preliminary Point Cloud Registration Test

Overview:

Goal: Validate the ability of the chosen algorithm to register the **simulated** acetabulum point cloud with the 3D scanned point cloud of the pelvis

Approach: Improve upon the ICP registration package offered by Open3D

Test 4:	
Preliminary Point Cloud Registration Test	
Objective	
Validate the selection of the registration algorithm for the use-case and test the ability of the registration algorithm to register the simulated fiducial points onto the pelvis point cloud.	
Equipment	MRSD Desktop 2, Atracys SpryTrack 300 Camera, Markers
Elements	Perception Subsystem
Personnel	Kaushik Balasundar
Location	NSH Basement
Procedure	
1. Load two pointsets, one of the complete pelvis model and the other of the downsampled point cloud from the surface of the acetabulum. 2. Run the selected registration algorithm and visually validate the transformation from pointcloud A to pointcloud B.	
Validation	
- Pointcloud B needs to be roughly overlapping with pointcloud A's acetabulum region to indicate that the registration has taken place.	

Test 4: Preliminary Point Cloud Registration Test

Approaches: Algorithms & Tools

- **Tools: Open3D** / ITK / PCL
  - Python-friendly
  - Prior experience
- **Algorithms: Iterative Closest Point** / Learning-based Methods
  - Native Open3D implementations & support
  - Industry standard for several years
  - Verified as a reliable method in other medical robotics applications

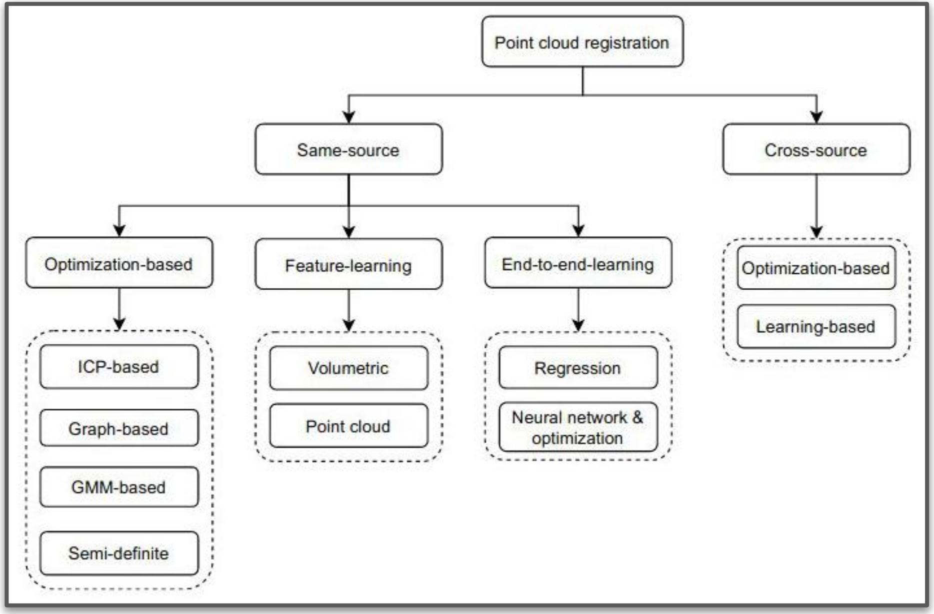


Figure: Registration Methods Overview

Test 4: Preliminary Point Cloud Registration Test

Approaches: Acquiring Test Model for Registration

- Off-the-shelf 3D model
- 3D scan model using Laser scanner (Konica Minolta Vivid 9i)
- 3D scan model using Camera / LiDAR setup (iPAD Pro / Kinect)

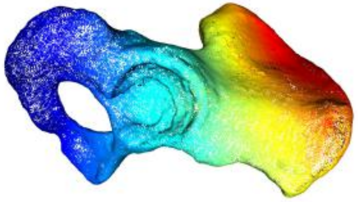


Figure: Off-the-shelf 3D model

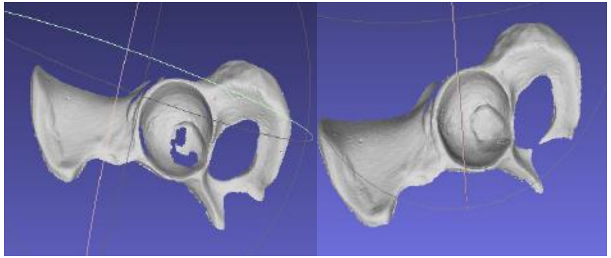


Figure: Konica Minolta Vivid 9i

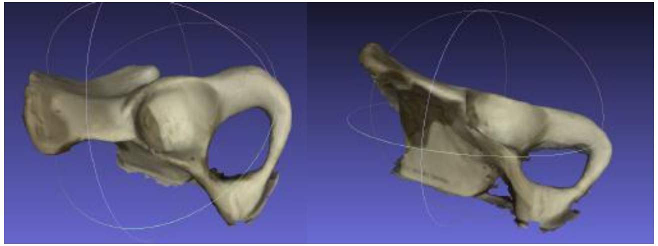


Figure: iPad Pro

Test 4: Preliminary Point Cloud Registration Test

Preliminary Experimentation

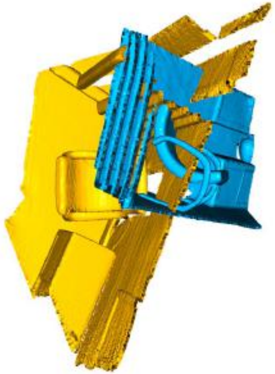


Figure: Two Pointclouds Initialized

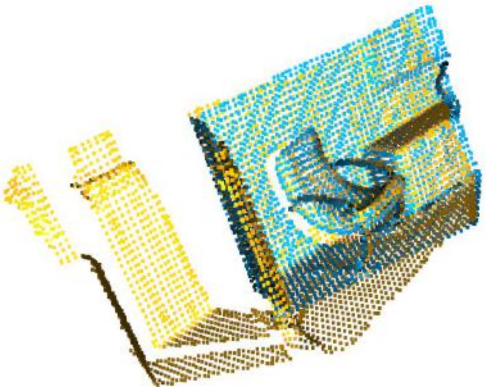


Figure: ICP Registration after Downsampling

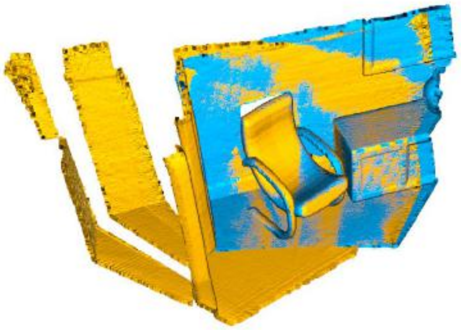


Figure: Result after RANSAC and upsampling

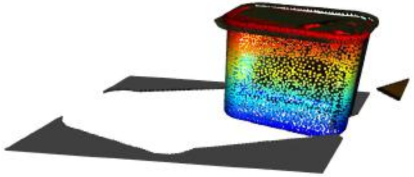


Figure: Custom Pointclouds

Test 4: Preliminary Point Cloud Registration Test

Validation

Validation
- Pointcloud B needs to be roughly overlapping with pointcloud A's acetabulum region to indicate that the registration has taken place.

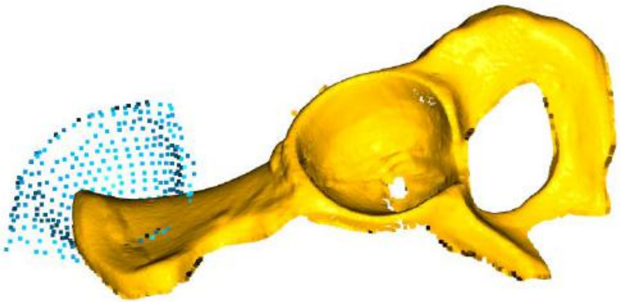


Figure: Source and target pointclouds

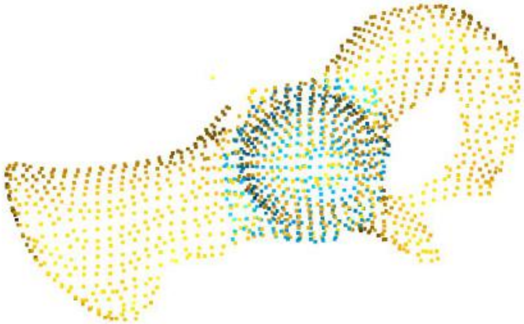


Figure: Downsampled pointclouds after initial registration

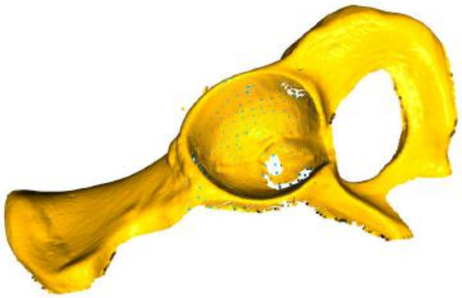


Figure: Point-to-point distance cost function results after RANSAC refinement

Test 4: Preliminary Point Cloud Registration Test

Challenges

- Post-processing 3D model from the scanner
  - Hole-filling using Autodesk MeshMixer
- Point cloud density differences
  - Source: 3D scanned pelvis (56704 points)
  - Target: Acetabular Surface (373 points)
- Hyperparameter tuning for registration and refinement
- RANSAC for fine-tuning post registration

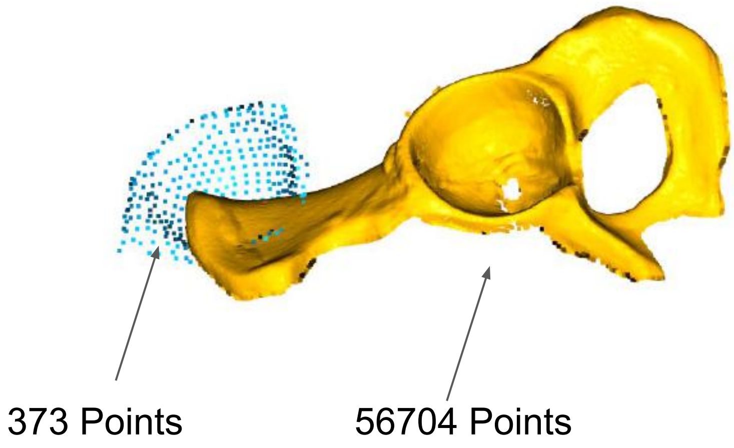


Figure: Density disparity between source and target pointclouds

Further Updates

Simulation Update

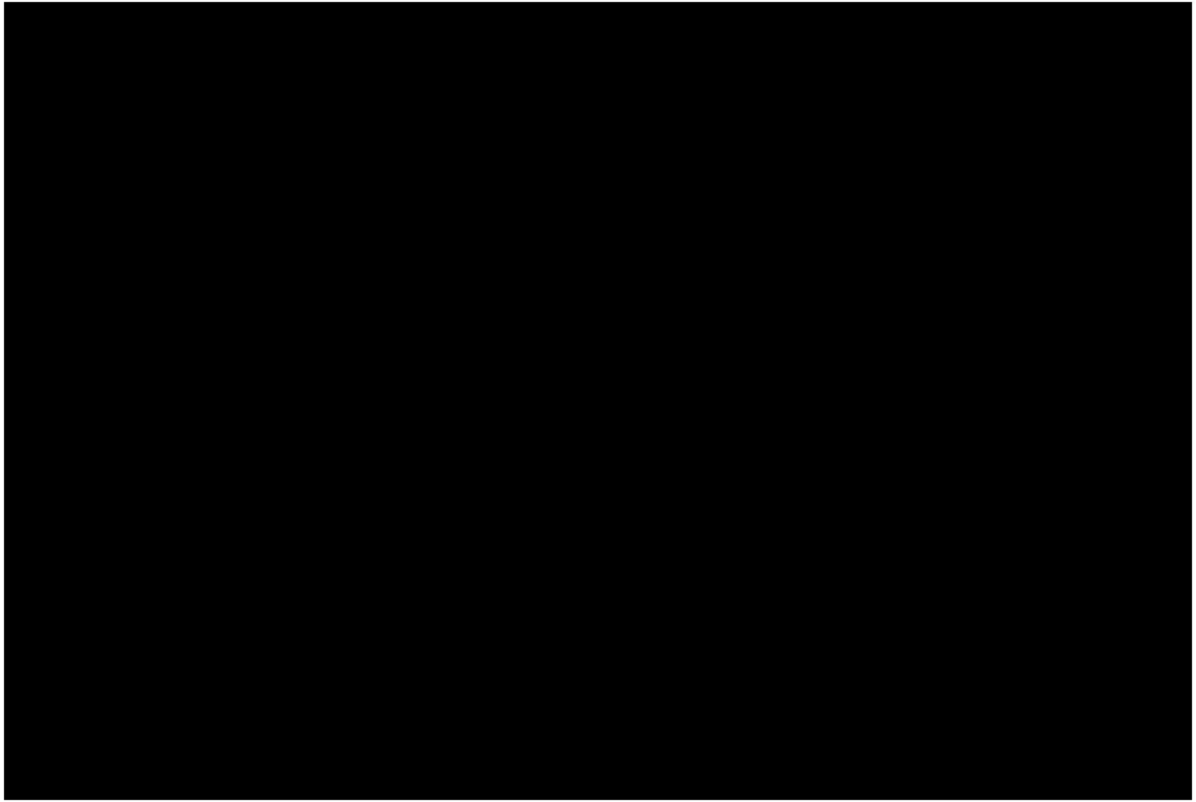


Figure: Simulation Environment on Gazebo & RViz

Controls Update: Tasks

- Updated Optimal Control Problem w/ Guidance from Professor Zachary Manchester
- Create Model and Dynamics in Julia using packages from the CMU Robot Exploration Lab
- Write Constraints and Objective function in Julia

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

$$\text{s.t.} \quad \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M^{-1}(\tau - \tau_{External} - C\dot{q} - G) \end{bmatrix} \qquad \text{State} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$$

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

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

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

$$q_k \epsilon q_{Limits}$$

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

Figure: Initial Optimal Control Formulation

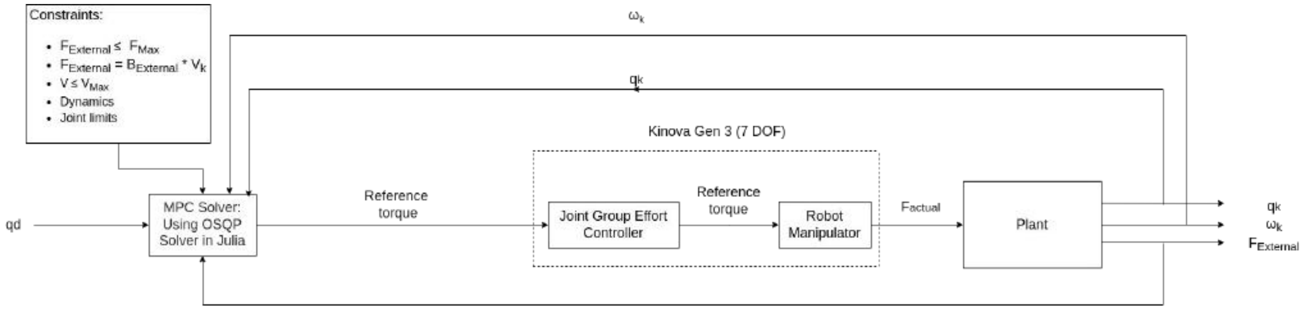


Figure: Control Architecture

Controls Update: Tasks

- Updated Optimal Control Problem w/ Guidance from Professor Zachary Manchester
- Create Model and Dynamics in Julia using packages from the CMU Robot Exploration Lab
- Write Constraints and Objective function in Julia

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

$$\text{s.t.} \quad \begin{bmatrix} \dot{q} \\ \ddot{q} \end{bmatrix} = \begin{bmatrix} \dot{q} \\ M^{-1}(\tau - \tau_{External} - C\dot{q} - G) \end{bmatrix} \qquad \text{State} = \begin{bmatrix} q \\ \dot{q} \end{bmatrix}$$

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

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

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

$$q_k \epsilon q_{Limits}$$

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

Figure: Initial Optimal Control Formulation

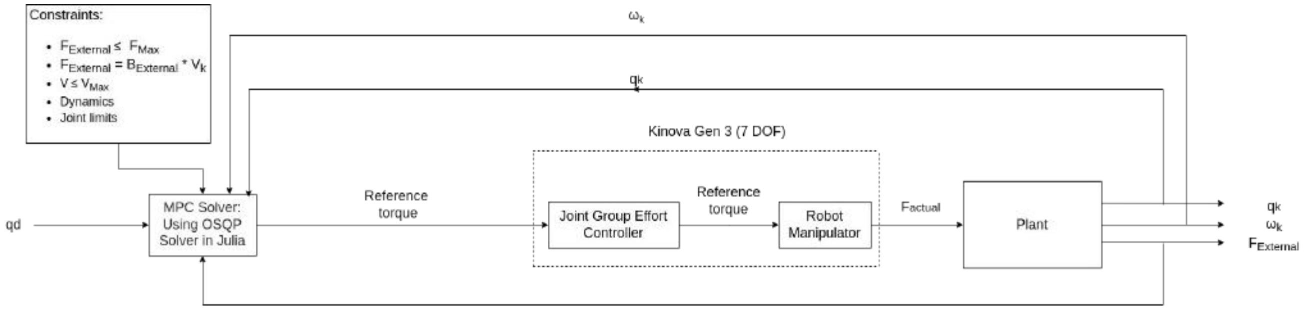


Figure: Control Architecture

Controls Update: Challenges

- Documentation is a little scattered:
  - Some libraries have dependencies on older versions of other libraries
- Even though packages have dynamics calculations, it doesn't provide an easy way to add external forces into consideration
- The way constraints are implemented in the packages makes it so that we have to keep track of extra variables within our state
  - Constraints must be directly from the state or control vectors
  - Those extra variables' dynamics also need to be calculated

```
# Calculate dynamics
RigidBodyDynamics.dynamics!(dynamicsResult, mechanismState, u)
# Add the effects of external forces/torques into dynamics
qdd = M\((M * dynamicsResult.v̇) - τ_ext)
```

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

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

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

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

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

$$q_k \epsilon q_{Limits}$$

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

Figure: Updated Optimal Control Formulation

Controls Update: Code

```
function RobotDynamics.dynamics(model::Arthur, x, 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 = x[1:num_q]
    qd = x[num_q+1:2*num_q]
    xd = x[2*num_q + 1:2*num_q + 6]
    F = x[2*num_q + 7:2*num_q + 12]
    Be = zeros(6, 6)
    if (norm(xd) > 1e-5)
        for k = 1:3
            Be[k,k] = norm(F) / norm(xd)
        end
    end

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

    # Get variables dependent on state
    J = getJacobian(model, q, qd)
    τ_ext = transpose(J)*Be*xd

    # Calculate dynamics
    RigidBodyDynamics.dynamics!(dynamicsResult, mechanismState, u)
    # Add the effects of external forces/torques into dynamics
    qdd = M\((M * dynamicsResult.v̇) - τ_ext)
    ẍ = getJ(model, J, qd, q)*qd + J*qdd
    Ḟ = Be*ẍ
    return [qd; qdd; ẍ; Ḟ; 0; 0; 0; 0; 0; 0]
end
```

Figure: Dynamics Implementation Code

```
# 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
ẋ_max = 0.0005 # m/s
vel_bnd = NormConstraint(n, m, ẋ_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
goal = GoalConstraint(xf)
add_constraint!(conSet, goal, N)
```

Figure: Problem Constraints Code

Hardware Update: Vention Table

Update

- Set up a Vention Table stand for our robot arm!

Challenges:

- Had to hand tap M8 holes into some of the Vention bars for connections to be properly made
- Spent large amount of time attaching the Vention together

Future Work:

- Get wooden base created for the bottom of the table for electrical components and storage
- Mount e-stop

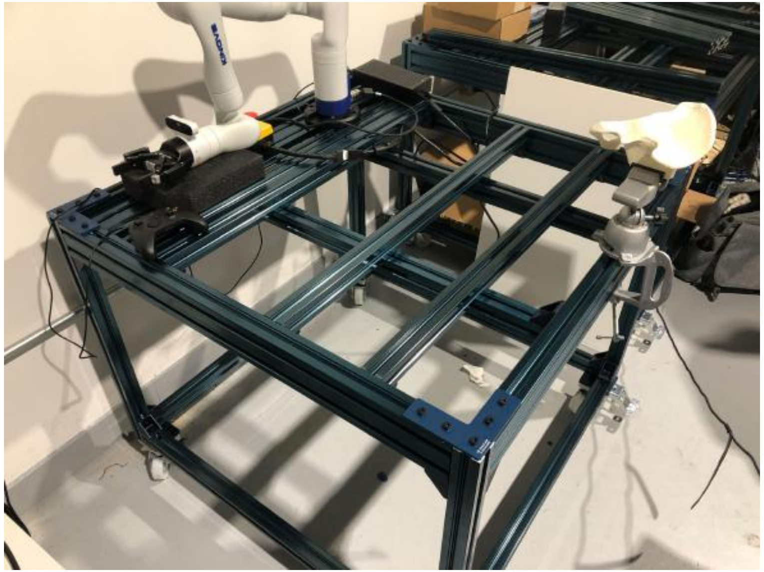


Figure: Vention Table Assembled

Hardware Update: Kinova Gen3 Arm

Update

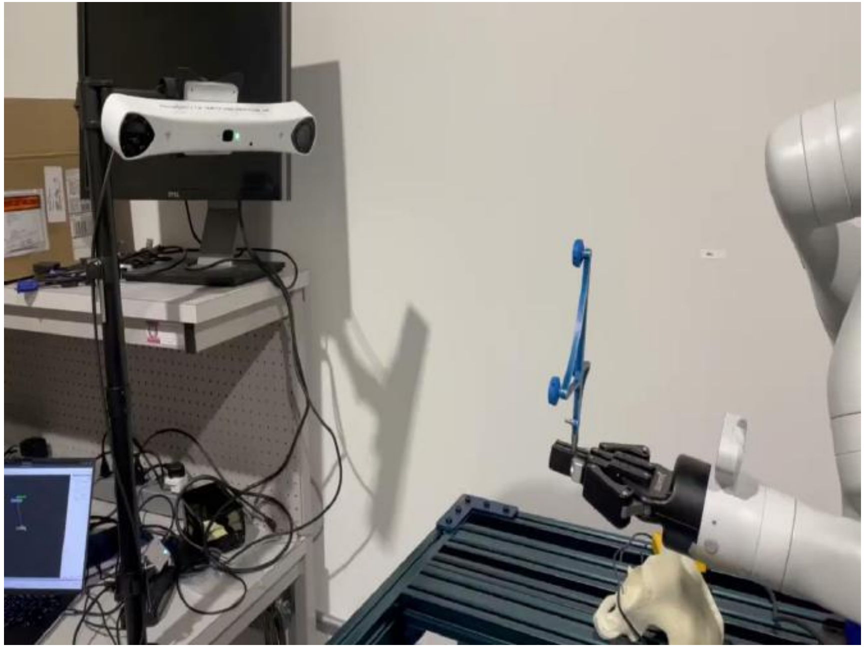
- Set up the Kinova Gen3 Arm!

Challenges:

- Had to exchange the other robot arm we previously had
- Needed to move around some of the Vention bars to fit the base, and it could still only fit sideways.

Future Work:

- Begin working on controlling the arm with ROS



Hardware Update: PCB Designed for End-Effector

Update

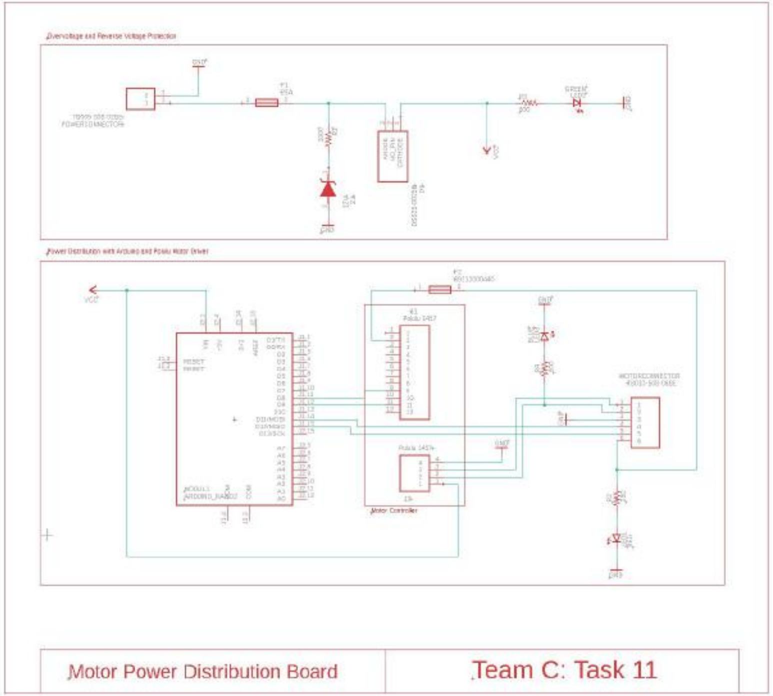
- Created a motor control PCB schematic for controlling a brushed DC motor for the acetabular reamer assembly

Challenges:

- Realized a power distribution system was less needed for our system and thus changed our PCB to be more of a motor control PCB
- Had some issues with creating libraries of custom parts

Future Work:

- Finalize parts and board layout
- Order parts for the end-effector



Hardware Update: End-Effector Redesign

Update

- Redesigned our end-effector based on feedback from sponsor
- Now going for a clamping design as seen on the right

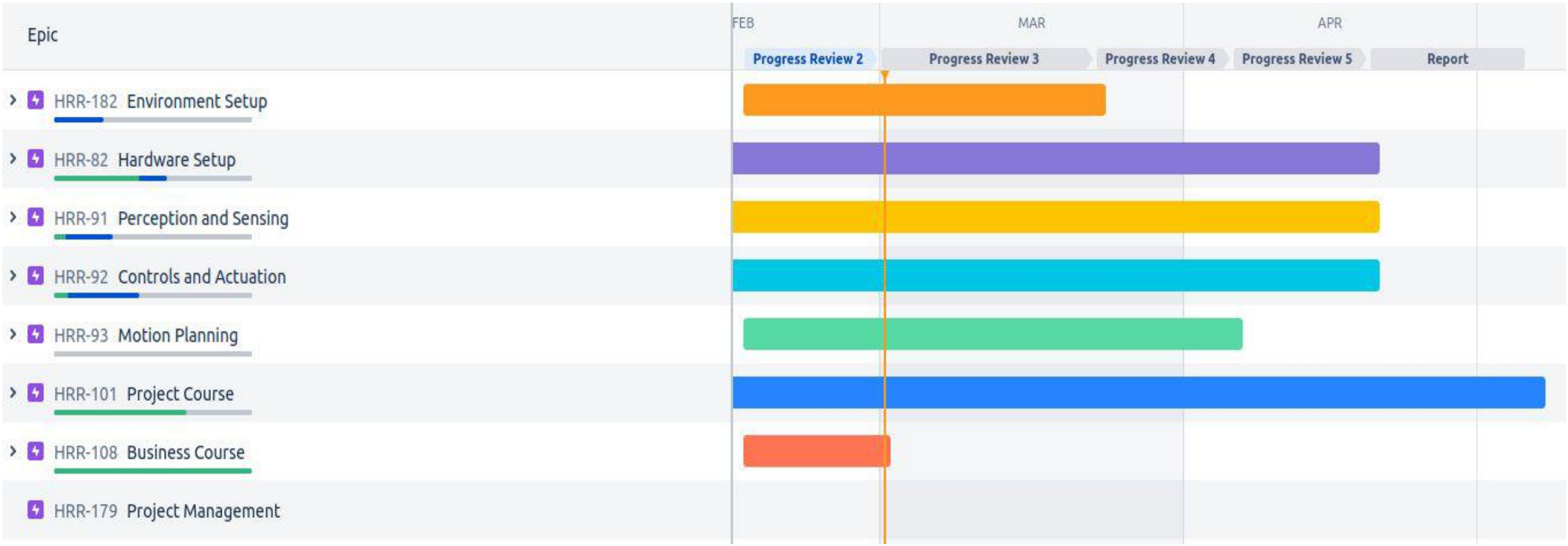
Challenges:

- 3D printed many similar designs with dimensional differences to find best fit

Future Work:

- Prototype final clamping design and look into rubbers that could be used with clamp
- Finalize end-effector design

Project Management Update: Updated Jira Roadmap



- Continue to work in 2-week sprints
- Hackathons on Fridays!

Plans : Progress Review 3

Progress Review #3 Tests

- ❑ Landmark Capture Perception and Sensing
- ❑ Waypoint/Trajectory Generation Planning and Controls
- ❑ Position and Force Control in Simulation Planning and Controls
- ❑ Reamer Motor Speed and Torque Hardware

Thank you!

Questions & Discussion