# Individual Lab Report - 3

Autonomous Reaming for Total Hip Replacement

HIPSTER | ARTHuR

Gunjan Sethi

Team C:

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

March 3 2022

Carnegie Mellon University
The Robotics Institute

## Contents

# 1 Individual Progress

## 1.1 Marker Re-calibration

In the previous sprint, the `camera_node` faced several reliability issues. After debugging the node and performing various tests with the help of the Atracsys SDK GUI, it was determined that the marker geometry file being used was inaccurate. Hence, a marker re-calibration was required in order to generate an updated version of the geometry file.
The re-calibration of the marker geometry was performed using the GUI. Once the new geomerty ini file was generated, it was loaded onto the GUI to test the marker detection robustness. The results were as expected within the error tolerance values. A video of the result was recorded that shows Anthony holding a registration probe. The geometry file loaded contains three fiducials. To test for robustness, a fourth fiducial is present on the probe. Despite adding the additional fiducial and various translation/rotational movements of the probe, the marker geometry is robustly detected. The video can be viewed here. The next task was to obtain marker poses via a ROS node.

## 1.2 Marker Pose Detection

As a continuation of the marker pose detection- preliminary tasks from the last report, this week's focus was on obtaining marker poses via ROS and publishing them onto a topic. The overall flow of tasks that the `camera_node` performs are shown in the Figure 1. The first two were part of the preliminary tasks performed last sprint.

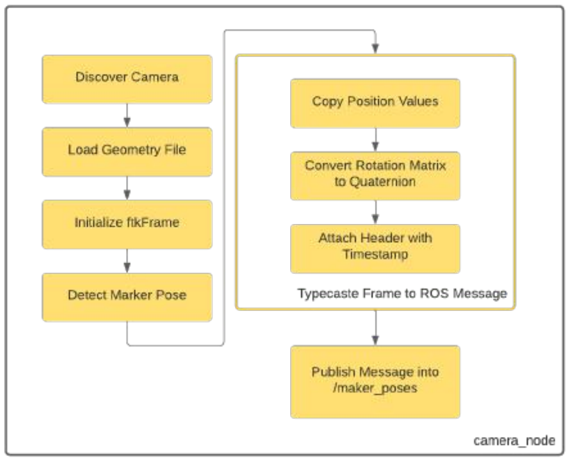

Discover Camera
Load Geometry File
Initialize ftkFrame
Detect Marker Pose
Copy Position Values
Convert Rotation Matrix to Quaternion
Attach Header with Timestamp
Typecaste Frame to ROS Message
Publish Message into /maker_poses
camera_node

**Figure 1:** Flowchart of Events in the `camera_node`

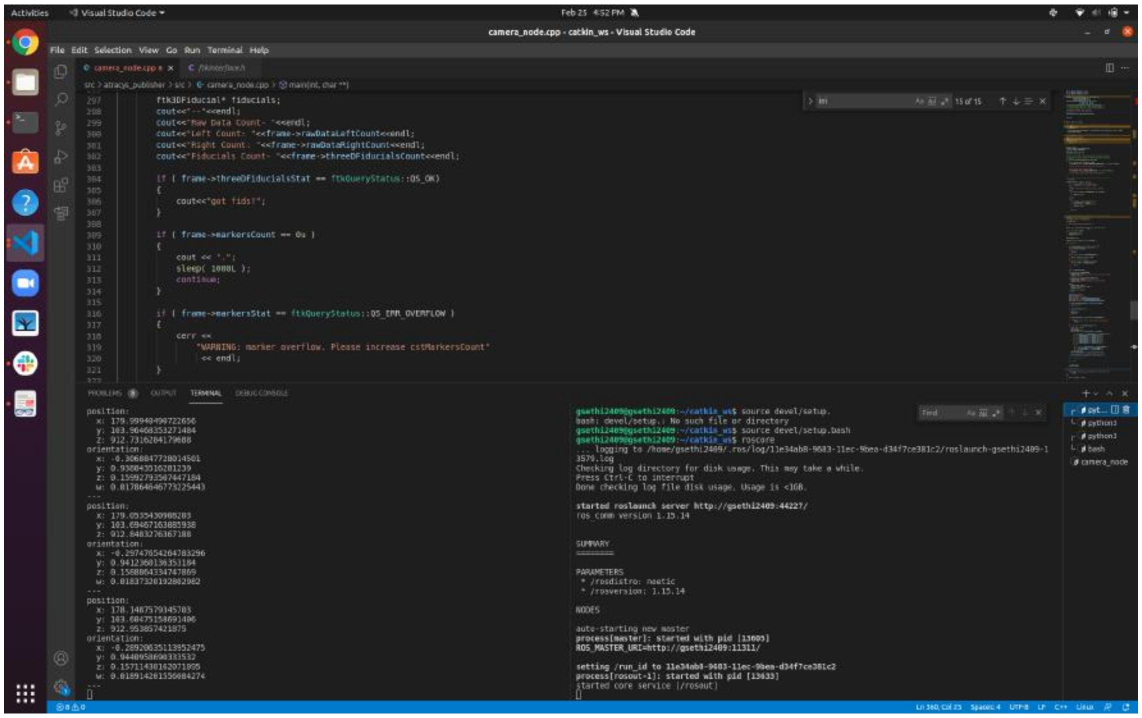
For this sprint, two major features, as explained below, were added to the `camera_node`.

### 1.2.1 Detecting

The objective of this task was to enable the `camera_node` to detect and print the number of fiducial points. Further, the node should identify the geometry of the fiducial points as a marker, based on the loaded geometry file. Finally it must print the 6DOF pose of the marker. The functions from the linked Atracsys SDK were used to initialize an `ftkFrameQuery` and detect the markers.

### 1.2.2 Publishing

Publishing the marker poses was challenging. In collaboration with Kaushik, the publishing functionality was developed and tested. For this, the marker poses were acquired. These marker poses are 6DoF with translation and a 3x3 rotation matrix. ROS messages use the quaternion representation. So, a conversion from rotation matrix to quaternion representation was performed using the Eigen library. Finally a header with timestamp was created for the pose and broadcasted. The publisher successfully was able to maintain the same FPS as the camera. Below is the screenshot of the published poses (Figure 2) and FPS measurements (Figure 3).


Visual Studio Code
camera_node.cpp - catkin_ws - Visual Studio Code
File Edit Selection View Go Run Terminal Help
camera_node.cpp
src > atracys_publisher > src > camera_node.cpp > main(int, char **)
PROBLEMS OUTPUT TERMINAL DEBUG CONSOLE

```
ftk3DFiducial* fiducials;
cout<<"---"<<endl;
cout<<"Raw Data Count- "<<endl;
cout<<"Left Count: "<<frame->rawDataLeftCount<<endl;
cout<<"Right Count: "<<frame->rawDataRightCount<<endl;
cout<<"Fiducials Count- "<<frame->threeDFiducialsCount<<endl;

if ( frame->threeDFiducialsStat == ftkQueryStatus::QS_OK)
{
    cout<<"got fids!";
}

if ( frame->markersCount == 0u )
{
    cout << ".";
    sleep( 1000L );
    continue;
}

if ( frame->markersStat == ftkQueryStatus::QS_ERR_OVERFLOW )
{
    cerr <<
        "WARNING: marker overflow. Please increase cstMarkersCount"
        << endl;
}
```

```
position:
  x: 179.99940490722656
  y: 103.96468353271484
  z: 912.7316284179688
orientation:
  x: -0.3068847728014501
  y: 0.930043516281239
  z: 0.15992793507447184
  w: 0.017864646773225443
---
position:
  x: 179.0535430908203
  y: 103.69467163085938
  z: 912.8483276367188
orientation:
  x: -0.29747654264783296
  y: 0.9412360136353184
  z: 0.1588864334747869
  w: 0.018373201928802982
---
position:
  x: 178.1487579345703
  y: 103.68475158691406
  z: 912.953857421875
orientation:
  x: -0.28920635113952475
  y: 0.9440958690333532
  z: 0.15711430162071895
  w: 0.018914281556084274
---
```

```
gsethi2409@gsethi2409:~/catkin_ws$ source devel/setup.
bash: devel/setup.: No such file or directory
gsethi2409@gsethi2409:~/catkin_ws$ source devel/setup.bash
gsethi2409@gsethi2409:~/catkin_ws$ roscore
... logging to /home/gsethi2409/.ros/log/11e34ab8-9683-11ec-9bea-d34f7ce381c2/roslaunch-gsethi2409-1
3579.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://gsethi2409:44227/
ros_comm version 1.15.14


SUMMARY
========

PARAMETERS
 * /rosdistro: noetic
 * /rosversion: 1.15.14

NODES

auto-starting new master
process[master]: started with pid [13605]
ROS_MASTER_URI=http://gsethi2409:11311/

setting /run_id to 11e34ab8-9683-11ec-9bea-d34f7ce381c2
process[rosout-1]: started with pid [13633]
started core service [/rosout]
```

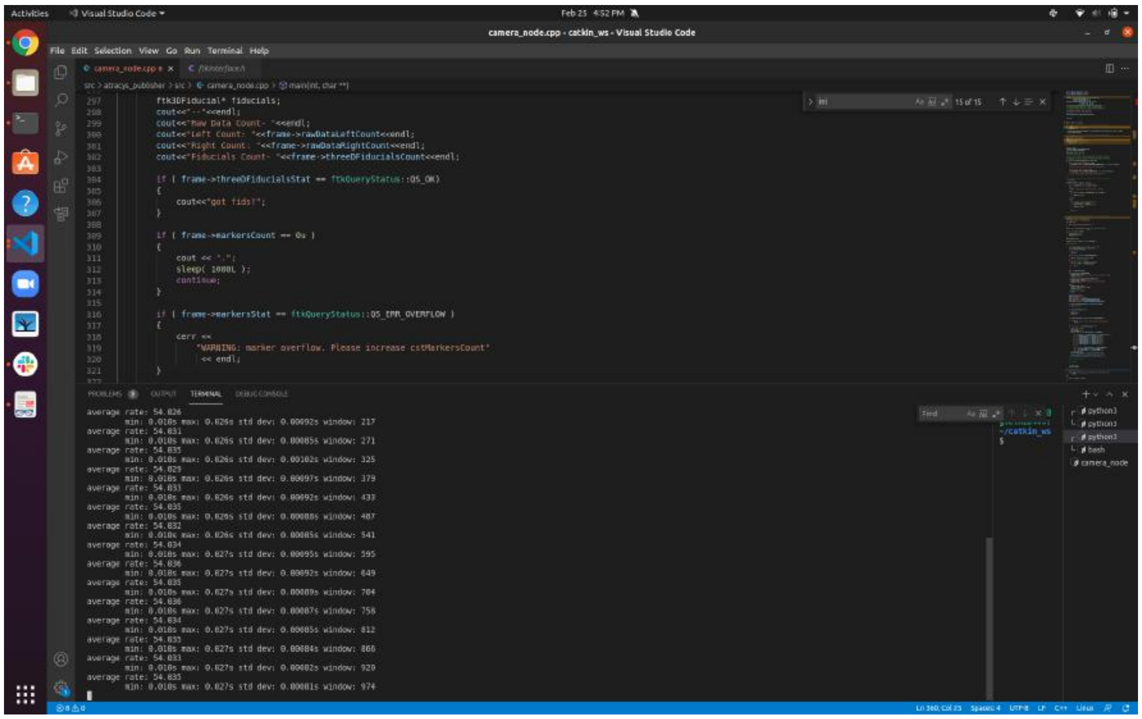
**Figure 2:** Published Marker Poses

### 1.2.3 Visualizing

RViz was used are the visualizer. Once the marker frame was typecasted and broadcasted, it was visible in RViz with respect to the camera frame. Figure 4 shows the visualization. Watch the visualization video complete video here.

### 1.2.4 Test

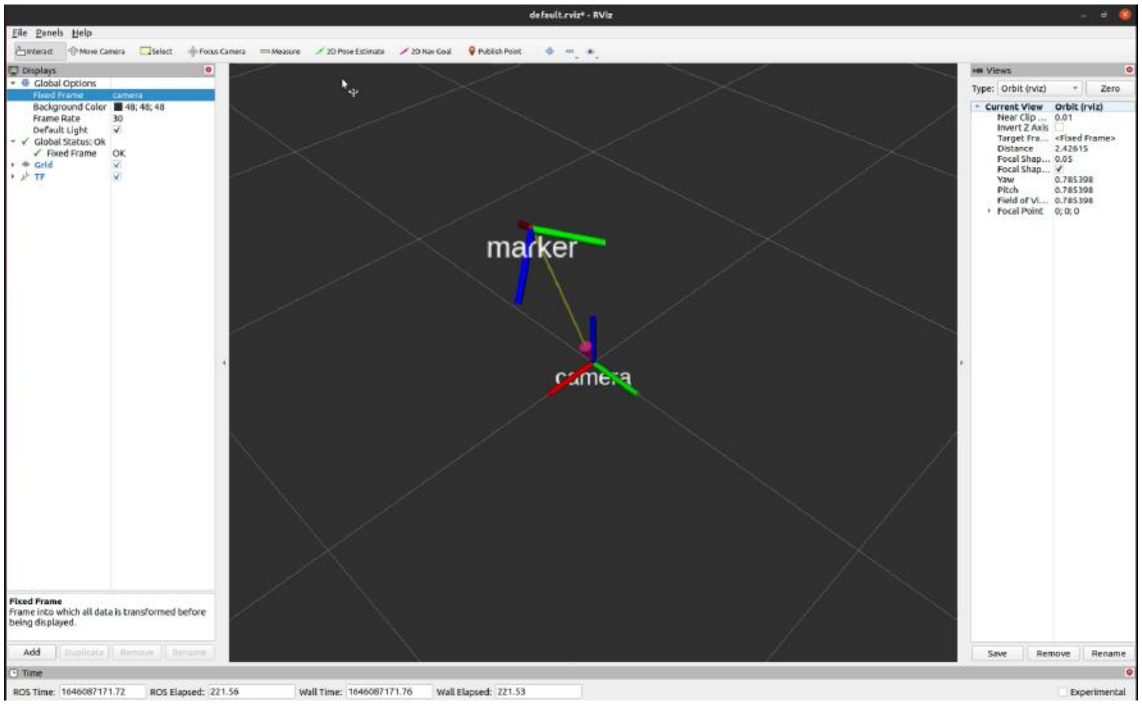
For the final testing, the robot arm was setup to grip onto the registration probe. The arm was controlled via the XBox and the robustness of the marker detection was validated. The test setup in Figure 5.


camera_node.cpp - catkin_ws - Visual Studio Code
src > atracsys_publisher > src > camera_node.cpp > main(int, char **)

```cpp
            ftk3DFiducial* fiducials;
            cout<<"---"<<endl;
            cout<<"Raw Data Count- "<<endl;
            cout<<"Left Count: "<<frame->rawDataLeftCount<<endl;
            cout<<"Right Count: "<<frame->rawDataRightCount<<endl;
            cout<<"Fiducials Count- "<<frame->threeDFiducialsCount<<endl;

            if ( frame->threeDFiducialsStat == ftkQueryStatus::QS_OK)
            {
                cout<<"got fids!";
            }

            if ( frame->markersCount == 0u )
            {
                cout << ".";
                sleep( 1000L );
                continue;
            }

            if ( frame->markersStat == ftkQueryStatus::QS_ERR_OVERFLOW )
            {
                cerr <<
                    "WARNING: marker overflow. Please increase cstMarkersCount"
                     << endl;
            }
```

```
average rate: 54.826
        min: 0.018s max: 0.026s std dev: 0.00092s window: 217
average rate: 54.831
        min: 0.018s max: 0.026s std dev: 0.00085s window: 271
average rate: 54.835
        min: 0.018s max: 0.026s std dev: 0.00102s window: 325
average rate: 54.829
        min: 0.018s max: 0.026s std dev: 0.00097s window: 379
average rate: 54.833
        min: 0.018s max: 0.026s std dev: 0.00092s window: 433
average rate: 54.835
        min: 0.018s max: 0.026s std dev: 0.00088s window: 487
average rate: 54.832
        min: 0.018s max: 0.026s std dev: 0.00085s window: 541
average rate: 54.834
        min: 0.018s max: 0.027s std dev: 0.00095s window: 595
average rate: 54.836
        min: 0.018s max: 0.027s std dev: 0.00092s window: 649
average rate: 54.835
        min: 0.018s max: 0.027s std dev: 0.00089s window: 704
average rate: 54.836
        min: 0.018s max: 0.027s std dev: 0.00087s window: 758
average rate: 54.834
        min: 0.018s max: 0.027s std dev: 0.00085s window: 812
average rate: 54.835
        min: 0.018s max: 0.027s std dev: 0.00084s window: 866
average rate: 54.833
        min: 0.018s max: 0.027s std dev: 0.00082s window: 920
average rate: 54.835
        min: 0.018s max: 0.027s std dev: 0.00081s window: 974
```

**Figure 3:** FPS measurements



| | |
|---|---|
| Global Options | |
| Fixed Frame | camera |
| Background Color | ■ 48; 48; 48 |
| Frame Rate | 30 |
| Default Light | ✓ |
| ✓ Global Status: Ok | |
| ✓ Fixed Frame | OK |
| Grid | ✓ |
| TF | ✓ |

**Fixed Frame**
Frame into which all data is transformed before being displayed.

Type: Orbit (rviz)

| | |
|---|---|
| Current View | Orbit (rviz) |
| Near Clip ... | 0.01 |
| Invert Z Axis | ☐ |
| Target Fra... | <Fixed Frame> |
| Distance | 2.42615 |
| Focal Shap... | 0.05 |
| Focal Shap... | ✓ |
| Yaw | 0.785398 |
| Pitch | 0.785398 |
| Field of Vi... | 0.785398 |
| Focal Point | 0; 0; 0 |

ROS Time: 1646087171.72 ROS Elapsed: 221.56 Wall Time: 1646087171.76 Wall Elapsed: 221.53

**Figure 4:** RViz Visualization

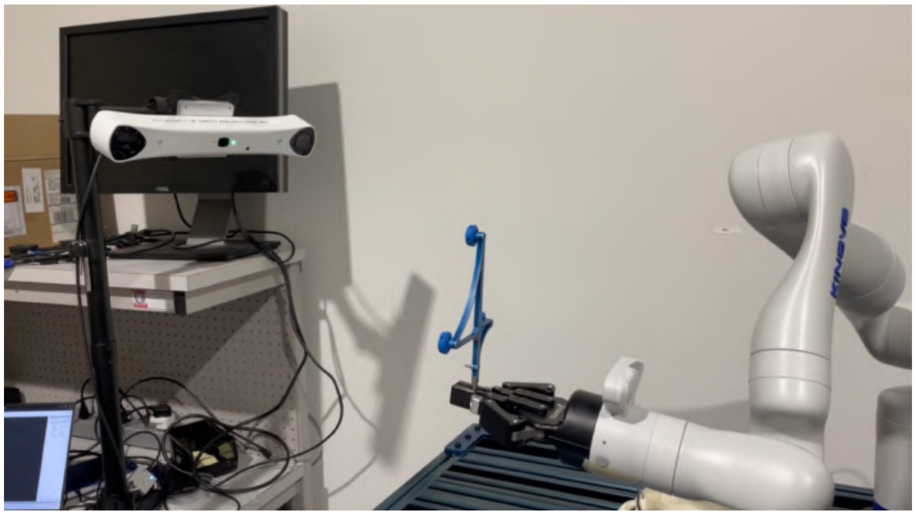


**Figure 5:** Test Setup

## 2 Challenges

### 2.1 Deprecated APIs and Missing Dependencies

While trying to import the tf library for transformations and broadcasting, various missing dependancies were found in the ROS installation. Reading carefully into the error messages and reading up on the ROS forum helped install the missing libraries.

### 2.2 Incorrect Units in Visualization

While visualizing the marker on RViz, the marker frame was not visible as it was too far from the camera frame. This is because the units were in meters (m) and needed to be converted to millimeters (mm).

## 3 Team Work

Following are the tasks accomplished by the team members since the previous ILR.

- ***Kaushik Balasundar*** 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.

- ***Parker Hill*** 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 the sponsors, and setting up the arm on the table. He designed and 3D printed prototypes for attaching the reamer handle to the end-effector. Further, he worked on the Power Distribution Board assignment, creating the conceptual design as well as the schematic of our motor control board.

- ***Anthony Kyu*** 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.

- ***Sundaram Seivur*** 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 model of the bone.

- ***Gunjan Sethi*** 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 future work, the following (individual) tasks have been planned for the MRSD project.

### 4.1 Setup MRSD Lab Desktop

All the development so far has been done locally on personal computers. For the upcoming Progress Reviews and demonstrations, the allocated MRSD Desktop will be utilized. This will require setting up the development environment with Ubuntu 20.04 and ROS Noetic with all other dependencies and testing if existing code runs smoothly on it.

### 4.2 Record Landmark PointCloud

- ***Generate Updated Geometry File*** The ROS node can now detect markers. However, to record landmark points on the pelvis, it is required to draw a geometrical relationship between the end of the registration probe and the fiducial markers. This might require caliberating a new marker geometry.

- ***Develop Function to Record Landmark Points*** Once a marker geometry has been calibrated, the next step would be to add a function into the `camera_node` to record and capture the landmark points.

- ***Publish and Visualize Landmark PointCloud*** Finally, the recorded points will be broadcasted and visualized on RViz. This would have its own complexities since the frame types from the camera would need to be typecasted into the PointCloud2 datatype from ROS.