Virtual Fixture for Surgical Robotics of Complex Geometry

Final Report

Computer Integrated Surgery II, Spring 2019

Members: Anurag Madan, Zhaoshuo Li
Mentors: Prof. Russell H. Taylor, Dr. Mahya Shahbazi, Dr. Niravkumar Patel

I. Introduction

In many surgical procedures, the patient's critical anatomies are close to the surgical field and exposed to the surgical tool. Any accidental contact with such anatomies can have dire consequences. An example of such risky surgical treatment is mastoidectomy, where the patient's' facial nerves are close to the drilling location. Contact with these nerves can cause significant nerve damage and paralysis. Therefore, this process requires high surgical skills and long hours of concentration, which can be tiring and difficult. Therefore, to ease the workload on surgeons and reduce the risk of surgery complication, it is desirable to create a forbidden area, so called *virtual fixture* around the anatomies, during robotic-assisted surgeries such that the surgeons can move around the surgical tool more "freely" and concentrate on other tasks better.

Virtual fixture has been proven to augment the surgeon's abilities. For example, in [1], the author explored the usage of virtual fixture to help guide the needle when performing suturing task, which increases the success rate. In [2], the author applied virtual fixture on neurosurgery to prevent vascular injury. Though the above examples were applied in different types of surgical robots, i.e., teleoperated versus cooperative, the general idea of adopting virtual fixture to constrain the user's motion is shared.

Though virtual fixture can bring benefits to robotic-assisted procedures, to have stable control behavior, the run loop frequency of the robot needs to be at least around 200 Hz for teleoperated systems and 100 Hz for cooperative systems. This limitation requires the virtual fixture computation to be relatively light. Imposing simple primitive constraints, such as planar, cubic, cylindrical and spherical constraints, can be computationally easy by using 3D geometry with known parameters. However, for a complex shape, such as the skull of a patient, needs to be approximated by several of the simple primitives, especially triangular meshes. If a naive implementation of examining every constraint during each run loop is used, the robot may become unstable due to the lag in the control loop. Therefore, suitable fast rejection strategy or proper data structure need to be adopted while implementing virtual fixtures.

The goal of this project is to develop virtual fixtures for relatively complex geometries while ensuring real-time performance for the Mark I robot from the Galen Robotics Inc.. The report is structured as following: Section II will give a detailed description of the project, including the software and hardware used; Section III will discuss the project goals; Section IV will discuss the technical approach the team has used; Section V and VI will discuss the experimental result and future work; Section VII will discuss the management summary.

II. Project description

A summary of the robot used for this project is given below:

- Hardware: The Galen Mark I is a surgical robot designed for hand-over-hand cooperative control for various surgical procedures, especially head and neck. This makes it an ideal robot for our project. The robot itself is a 5DOF, parallel-link robot, having a translational delta stage in X, Y, and Z directions, and a roll and tilt rotational stage. The 6th rotation axis was not added as most surgical tools do not need to be spun about their axis, and those that do have their own spinning mechanism independent of the motion and control of the robot.

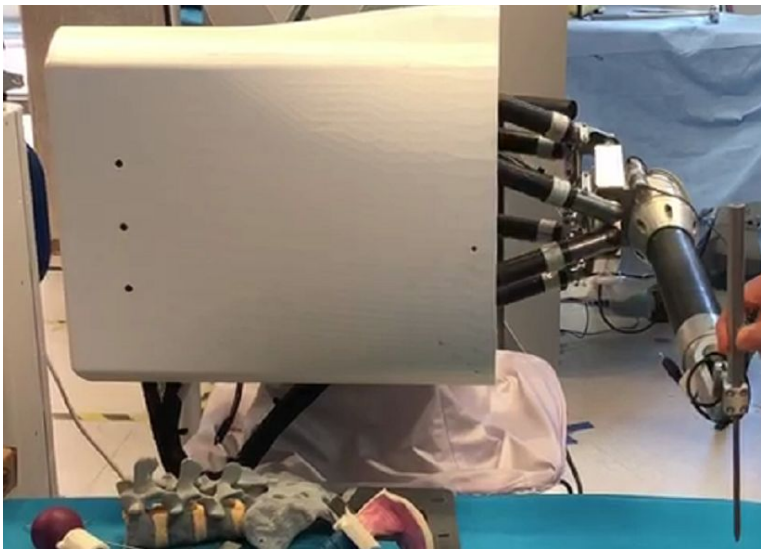

Fig 1. Galen Mark I robot

- Software: The software used to develop the Mark I robot is based extensively on the open-source CISST-SAW Library [3], developed at the Laboratory of Computation and Sensing Robotics (LCSR), Johns Hopkins University. The library supports many interfaces with different hardwares, such as ATI force sensor. The library also provides numerical computation packages and high level functions for multi-process program development.

III. Project goals

The goals of this project were to design and implement simple and complex virtual fixtures for the Galen Mark I robot. A selection of virtual fixtures was identified that gave the most versatility in terms of guidance to the surgeon. The deliverables for the project is summarized below:

- **Minimum deliverable**: A software framework for simple virtual fixtures, including plane constraints and insertion along an axis.
- **Expected deliverable**: A software framework for more complex virtual fixtures like constraints for 3D surfaces. The framework should also be able to automatically switch modes from virtual fixtures to free motion depending on location of the robot tool tip.
- **Maximum deliverable**: A software framework to build mesh files from CT scan data, register the physical phantom, and compute virtual fixture constraints.

IV. Technical Approach

Constraint Optimization

The control of the robot, along with virtual fixture constraints, can be formulated as an optimization problem. For a hand-over-hand robot like the Galen Mark I, the user input is converted to the motion of a robot by the following objective

$$argmin\ |J\Delta q - \tau|$$

where J is the Jacobian of the robot, Δq is the incremental joint position (or joint velocity) of the robot within one iteration, and τ is the force/torque input of the user. The virtual fixtures can be imposed as additional inequality constraints as following.

- **Insertion along Axis** - This fixture allows the user to insert the tool along the tool axis, without having any deviation, which is useful for drilling tasks. This is equivalent to locking the rotational axis of the robot, which can be achieved by

  $$H\Delta q \leq 0$$

  where H is a $6 \times N$ matrix, with N being the number of joints of the robot. The H is constructed such that only Cartesian rotational movement will be left. For Mark I,

  $$H = \begin{matrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \end{matrix}$$

and $N = 5$.

- **Single Plane** - This was an intermediate fixture, which creates a simple plane constraint with its normal along the tool axis, and with a specified offset. The robot is then constrained to remain above or below the generated virtual plane. Single Plane Constraint was also utilized in implementing subsequently more complex virtual fixtures. Single Plane Constraint is a primitive inequality constraint, which is formulated as:

$$N \cdot J\Delta q \leq d/(\Delta t)$$

Where N is the unit normal to the plane, J is the Jacobian of the robot, d is the distance from the robot tip to the plane, and Δt is the period of the robot run loop. During our implementation for 6 DoF Single Plane Constraint, the team had to reduce the gain of the rotational joints to increase the stability. This is because the optimizer will tries to move rotational joints to satisfy the above inequality constraint, rather than stopping the translational joints, for which the latter is more desirable. When the speed of the robot approaching the plane is high, the rotational joints may have jerky motion.

- **Multi Plane** - This constraint breaks a concave surface into a specified number of planes, and implements a plane constraint (discussed above) for all the obtained planes. For the demonstration, a cylinder was broken into 500 planes, with the tooltip inside the cylinder. This constraint was designed for concave surfaces. Multi plane constraint, and subsequently, multi mesh constraint, were implemented by locking rotation of the robot. This is because locking the orientation of the robot allows the team to solve a relatively simpler problem. Moreover, locking the orientation is also effectively equivalent to solving the constraint optimization problem in configuration space, thus having the value of transporting the code base directly.
Multi Plane constraint was implemented by constraining the robot between multiple planes by having an inequality constraint for each plane, thereby generating ‘n’ inequality constraints for ‘n’ planes.

$$-N_1 \cdot J\Delta q \leq d_1/(\Delta t)$$
$$-N_2 \cdot J\Delta q \leq d_2/(\Delta t)$$
$$\dots$$
$$-N_n \cdot J\Delta q \leq d_n/(\Delta t)$$

Where $N_1, N_2, \dots N_n$ are the unit normals to the planes, J is the Jacobian of the robot, $d_1, d_2, \dots d_n$ are the distances from the planes to the robot tip, and Δt is the time step of the robot run loop.
This constraint works well when the surface is concave. However, this kind of constraint fails when the region is convex. As can be seen from fig. 2, the red regions cannot be optimized, as the robot tooltip is below a plane, thereby violating the inequality constraints. It can also be seen that plane constraints divide the workspace into multiple parts, as in fig 2, the workspace gets divided into 7 parts (R1 - R7), and only on the concave side of the surface is the entire region accessible (R1). On the convex side, it is not possible to go from one region to another (say from R3 to R4, due to a plane constraint).

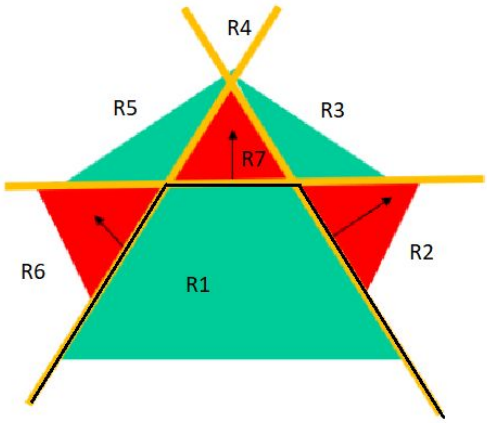

Fig 2. Multi-plane constraint fails for a convex shape. Orange lines: planes; Black lines: surface; Arrows: normal direction; Green regions: optimization is feasible; Red regions: optimization fails.

- **Multi Mesh** - This constraint creates a surface from a given mesh. Compared to the plane constraint, the triangle mesh will have a finite area, i.e. not extending to infinity. Therefore, it is more suitable to approximate a concave shape compared to multi plane. The inequality constraint will bound the distance travelled by the tooltip to be less than the distance to the closest points on the meshes, thus *closest point constraint* was implemented. For the demonstration, a simple cube mesh was used, with the tooltip initialized outside the cube. The inequality constraints for multiple planes are

$$-N_1 \cdot J\Delta q \leq d_1 /(\Delta t)$$
$$-N_2 \cdot J\Delta q \leq d_2 /(\Delta t)$$
$$\dots$$
$$-N_n \cdot J\Delta q \leq d_n /(\Delta t)$$

  where N_i is the unit normal direction of the i-th mesh, J is the Jacobian of the robot, d_i is the distance of robot tool tip to the closest point on the given mesh, and Δt is the time step of the robot run loop. However, for a concave shape, not all closest points should be included, especially for the cases where closest points are on edges/corners. Three cases were taken into account
  - **Closest point is within the triangle**: in this case, the constraint is equivalent to a plane constraint, and therefore the normal of the triangle mesh is used directly

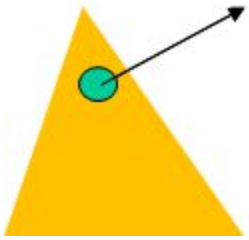

Fig 3. Triangle mesh constraint when closest point is within the triangle. Orange triangle: meshes; Solid arrows: final direction; Green dot: closest points.

- **Closest point is on the edge/corner, and only found once**: in this case, the robot is closest to the edge/corner of a single triangle mesh. That edge, in a concave case, will not affect the behaviour of the robot at all. Therefore, it is discarded during the optimization.

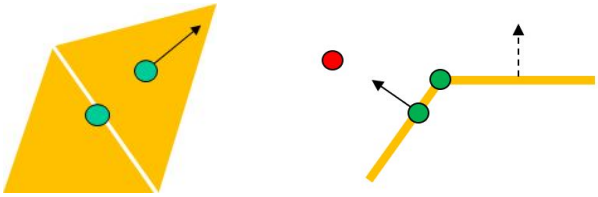

Fig 4. Triangle mesh constraint when closest point is on the edge/corner of a triangle. That point is discarded for the constraint optimization. Orange triangle: meshes; Dashed arrows: normal direction of mesh; Solid arrows: final normal direction used for constraint optimization; Green dot: closest points; Red dot: robot tool tip.

However, this will fail for a concave case, since robot can potentially go beyond the constrained region if the edge point is rejected as before, which is shown as below

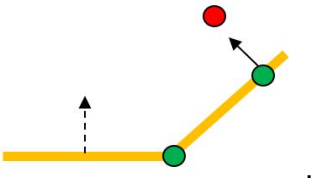

Fig 5. Failure case when the closest point is discard for a concave shape. Orange triangle: meshes; Dashed arrows: normal direction of mesh; Solid arrows: final normal direction; Green dot: closest points; Red dot: robot tool tip.

- **Closest point is on the edge/corner, and found multiple times**: in this case, the robot is closest to the edge/corner of multiple triangle meshes. In this case, the averaged normal from different meshes is used to best represent the constraint.

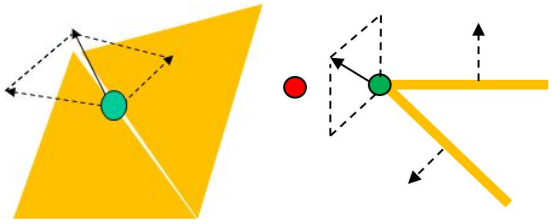

Fig 6. Triangle mesh constraint when closest point is on the shared edge/corner of triangles. The normal of that point is approximated as the average of adjacent meshes. Orange triangle: meshes; Dashed arrows: normal direction of mesh; Solid arrows: final normal direction used for constraint optimization; Green dot: closest points; Red dot: robot tool tip.

- **Error recovery**: Since there is inherent error in the robot kinematics, in very rare conditions, for example the edge of the meshes, the robot can get inside the forbidden area and failed the optimizer. Therefore, the team formulated an error recovery strategy. The idea being that having robot moving in the normal direction of the mesh/plane such that it will be on the correct side of the plane as soon as possible, without violating the maximum velocity that it can travel. When the robot accidentally enter a predefined shape, the inequality constraint changes to

  $$N \cdot J\Delta q \geq min(d_{max},\ d)/(\Delta t)$$

  where $d_{max} = v_{max}\Delta t$ is the maximum distance the robot can travel in one iteration, N is the unit normal direction of the mesh, J is the Jacobian of the robot, d is the signed distance of robot tool tip to the closest point on the given mesh, and Δt is the time step of the robot run loop. The detection of this case is easily found by checking

  $$N \cdot (p - c) < 0$$

  where p is the current position of the robot, c is the closest point on meesh.

Covariance tree

To reduce the computation burden of solving constraints, not all of the constraints should be added to the optimization solver. Therefore, a bounding box is built around the robot tooltip with the bounding box edge length to be the maximum distance the robot can travel in one iteration. For multi plane case, if the bounding box intersects with a plane, it is added to the solver. Otherwise, it is discarded in the current iteration.

For the multi mesh constraint case, a covariance tree [4] is built to further speed up the search. Each node within the covariance tree is bounded with a box, whose edges are aligned with the major covariance direction within the node (see Fig 7). When an intersection is found between a node and robot tool tip, intersection detection is again performed on the child nodes. When a leaf node is reached, the closest points on each triangle mesh contained in the node are returned and also flags indicating if they are on edge/corner.

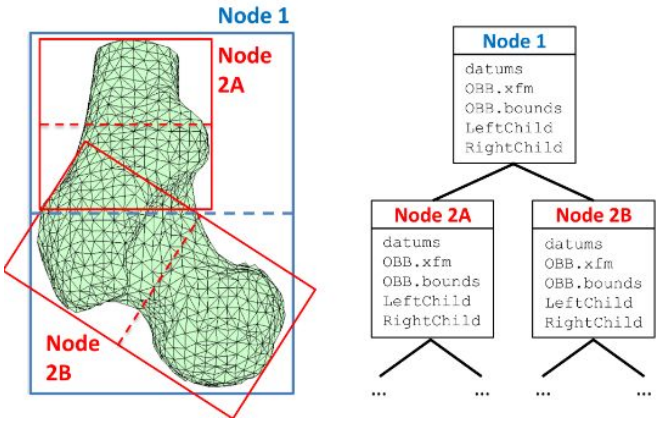

Fig 7. Illustration of nodes forming the upper two levels of a standard PD tree [5].

V. Results

By the time of submission of this report, only the minimum and expected deliverable has been met. During the experiment, all virtual fixtures implemented are qualitatively evaluated. It has shown that the simple virtual fixture such as insertion along axis and 3 DoF single plane constraint works properly. As discussed earlier, the 3DoF single plane constraint works as intended, while 6 DoF single plane constraint has some jerky motion on the rotational joints. As for multi plane constraint for concave surface, the functionality works and run time periods are recorded for different number of active planes. As for multi mesh constraint for convex surface, the robot is constrained properly. The experiment result of run time period versus number of active constraints are shown in plot below. In each case, the number of total constraints are 500, 1000, 1500, 2000, 2500, 5000 and 10000 planes.

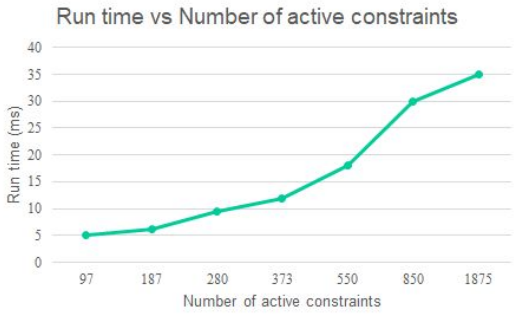

Fig 8. Run time versus Number of active constraints

VI. Future Work

As discussed in the previous sections, there are three future directions that this work needs to be carried on:

- Formulating a better plane constraint strategy for 6 DoF case to reduce the jerky motion of the rotational joints and fully stop the motion on a plane.

- Formulating a better mesh constraint for edge/corner case to prevent entry of both convex and concave shapes.

Zhaoshuo Li will continue to work on this project in the Fall 2019 term to extend the algorithm as discussed above.

VII. Management Summary

Anuarg Madan was mainly in charge of:

- Constrained optimization formulation for plane constraint
- Testing

Zhaoshuo Li was mainly in charge of:

- Constrained optimization formulation for point constraint
- Covariance tree
- Testing

The software framework for Galen Mark I is currently stored on BitBucket server (http://bitbucket.org), *researchrepo* repo, *mk1-vf* branch.

The software framework for covariance tree is currently store on GitLab LCSR server (https://git.lcsr.jhu.edu), *PDTree* repo, *devel* branch.

All necessary documentations are uploaded to CIS II project page (https://ciis.lcsr.jhu.edu/dokuwiki/doku.php?id=courses:456).

References

[1] A. Kapoor. Motion Constrained Control of Robots for Dexterous Surgical Tasks. PhD thesis, Johns Hopkins University, September 2007.

[2] Xia, Tian. Model Driven Robotic Assistance for Human-Robot Collaboration. Diss. Johns Hopkins University, 2013.

[3] Jung, M., et al. "A surgical assistant workstation (saw) application for teleoperated surgical robot system." The MIDAS Journal-Systems and Architectures for Computer Assisted Interventions (2009).

[4] J. P. Williams, R. H. Taylor, and L. B. Wolff, “Augmented KD techniques for accelerated registration and distance measurement of surfaces,” in Computer Aided Surgery: Computer-Integrated Surgery of the Head and Spine, Sep. 1997, pp. 1–21

[5] Billings, Seth D. Probabilistic Feature-Based Registration for Interventional Medicine. Diss. Johns Hopkins University, 2015.