Image-guided Robotic Navigation Final Report

Bangsheng Jiang

1. Introduction

Computer-aided image processing techniques have found widespread application in various medical practices, enhancing accuracy, convenience, and objectivity. Among these techniques, 3D reconstruction stands out, while others automate tasks traditionally performed by radiologists, such as identifying different tissues[1,2] or calculating specific values for diagnosis and disease staging[3,4,5].

In recent years, there is growing evidence that image processing plays a crucial role in surgical procedures. For instance, intra-operative imaging has the potential to maximize glioma resection during surgery[6]. Additionally, computer-assisted navigation has been shown to improve patient outcomes in total hip arthroplasty, reducing the risk of dislocation and sepsis[7,8,9]. It's important to note that this list is not exhaustive, as this technology continues to find applications across a broader spectrum of medical fields.

Digital twins (DT) technology can be a convergence of image processing and robot simulation. DT are virtual representations that precisely reflect their physical counterpart. In surgical contexts, this is creating 3D models that depict patients' unique anatomical structure derived from imaging data. Although the concept of DT is relatively recent, similar practices have a longstanding history in medicine. For instance, virtual bronchoscopy, introduced in 1995[10], utilizes non-contrast-enhanced CT images to render a patient's lung airways within a virtual environment. While being a non-invasive technique and can not perform surgical procedures, it serves as an invaluable tool for assessment[11] or as a guide during actual bronchoscopic interventions[12,13]. A review indicates that DT offers clinicians the chance to rehearse surgical procedures in a simulated environment[14], decreasing complication rates and the steepness of learning curves[15].

In contrast to traditional surgery, where surgeons have direct physical interaction with the patient, robotic surgery involves the surgeon operating through a console, which can inherently serve as an interface to the digital space[15]. Consequently, there is no necessity to construct additional virtual reality (VR) or augmented reality (AR) simulation systems, simplifying the process of simulating robotic surgery on DT.

DT technology not only enhances patient outcomes but also plays an instrumental role in refining the product development cycle by facilitating a comprehensive end-to-end pipeline. Currently, the lifecycle data of most products are centered around physical items, leading to heterogenous data that is often "isolated, fragmented, and stagnant"[16], presenting integration challenges and hindering the creation of a seamless end-to-end pipeline. DT and other simulation technologies act as a nexus, integrating these varied data sources to yield valuable insights during product implementation and validation.

In the design phase, akin to other industries, simulation enables designers to identify and promptly rectify product flaws before advancing to subsequent stages. More importantly, considering the inherent challenges of human clinical trials—which are time-consuming, costly, and subject to strict ethical and legal regulations—virtual verification emerges as an exceptionally useful tool for accelerating the product development cycle within the healthcare industry.

2. Method

This research project aims to identify the optimal trajectory between two sets of points that minimizes the risk of encountering critical structures. The path planning algorithm takes in 4 point lists as input, including entry and target candidates, as well as critical structures, which are vessels and ventricles. The algorithm returns the optimal entry and target points of the trajectory. The user can specify a maximum trajectory length. If the search fails, the program will provide an error message. With these 2 points, user can create a *vtkMRMLLinearTransformNode* that represent the movement of the needle. Subsequently, it will be sent to an Ubuntu Virtual Machine via the OpenIGTLinkIF module in 3D Slicer[17].

The robotic simulation is carried out using ROS 2. First, an .xml file describing a surgical robot is created following the Unified Robot Description Format (URDF)[18]. MoveIt Setup Assistant[19] is employed to integrate this robot into the MoveIt pipeline. The robot's movement planning and simulation are then facilitated by the Move Group Python Interface, which takes input from the transformation and user-defined variables[24]. Finally, users can see robot's movement in RViz.

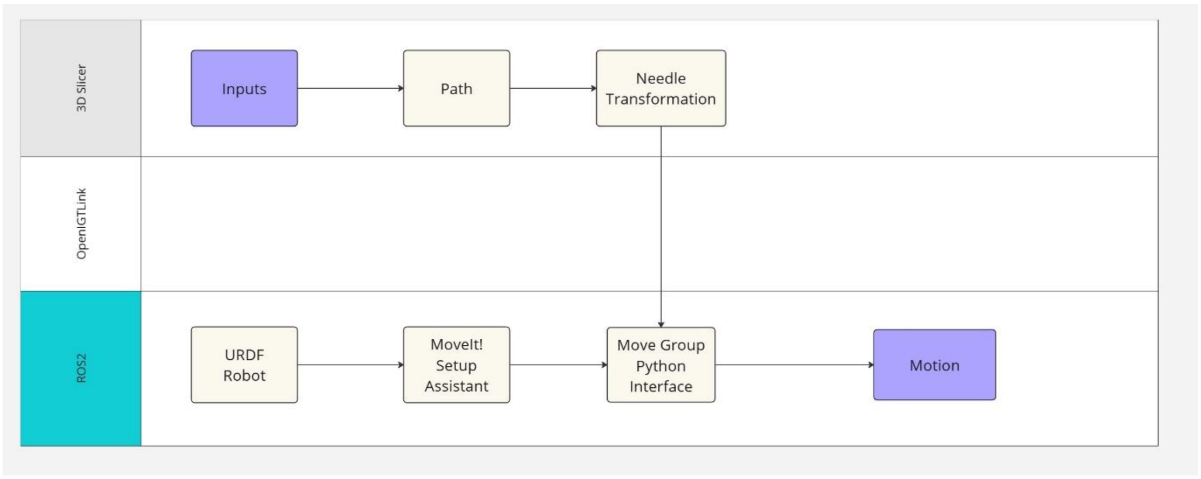


Figure 1: Overview of Workflow

2.1 Path Planning in 3D Slicer

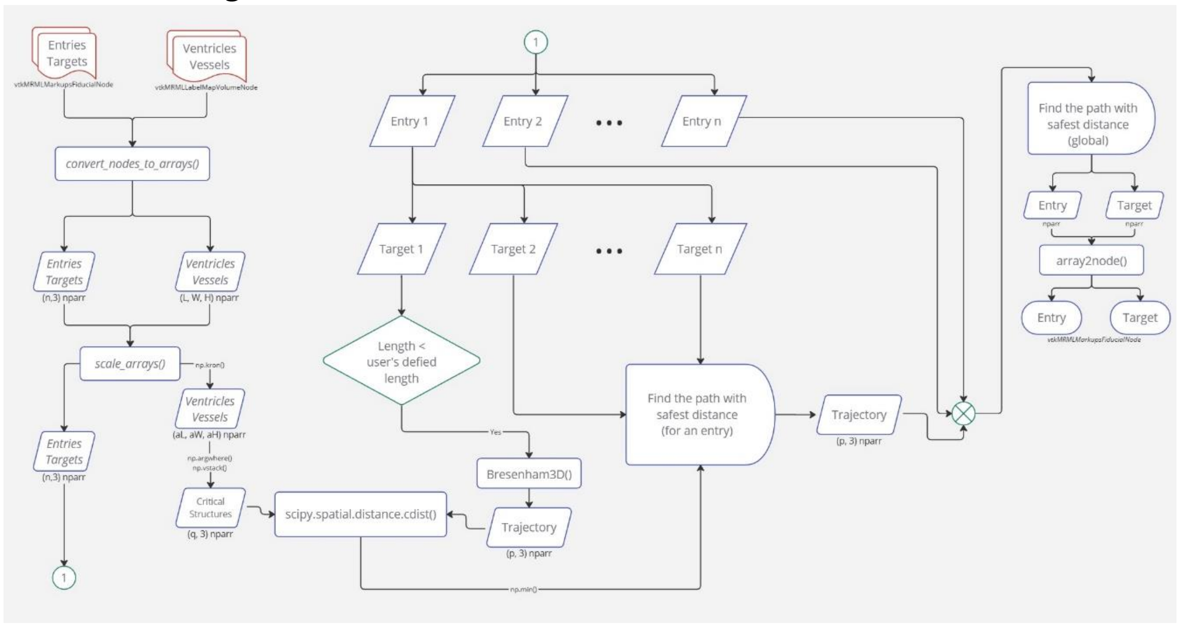


Figure 2: Flowchart for path planning algorithm

The flowchart for this algorithm is shown as Fig.2. Entry and target candidates are represented by .fcsv files that are read as vtkMRMLMarkupsFiducialNode in 3D Slicer. Critical structures, which include ventricles and vessels, are stored as .nii.gz files and are read as vtkMRMLLabelMapVolumeNode. The pathfinding algorithm will yield two points, both represented as vtkMRMLMarkupsFiducialNode: the optimum entry point and the target point.

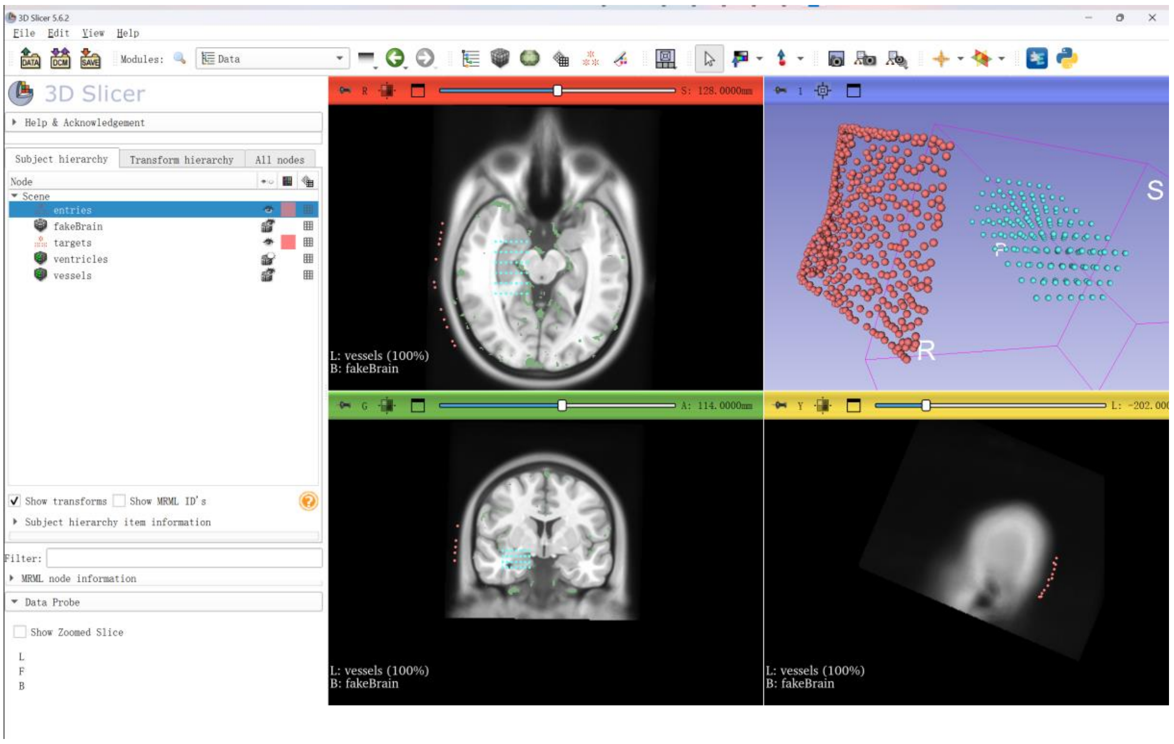


Figure 3: screen shot for entries(red), target(cyan), and critical structures(green, only in cross-sections) in a fakebrain

The algorithm employs a brute force approach and is implemented in Python. It transforms vtkNode objects into NumPy arrays and utilizes Bresenham's algorithm to compute the trajectory from the entry to the target point. Subsequently, it verifies if the trajectory length remains within the user-defined maximum threshold. If affirmative, the algorithm proceeds to calculate the distance between the trajectory and critical structures, identifying the path that maintains the safest (maximum) distance. Finally, it converts the coordinates of the optimal path's starting and ending points back into *vtkMRMLMarkupsFiducialNode* format for visualization in 3D Slicer.

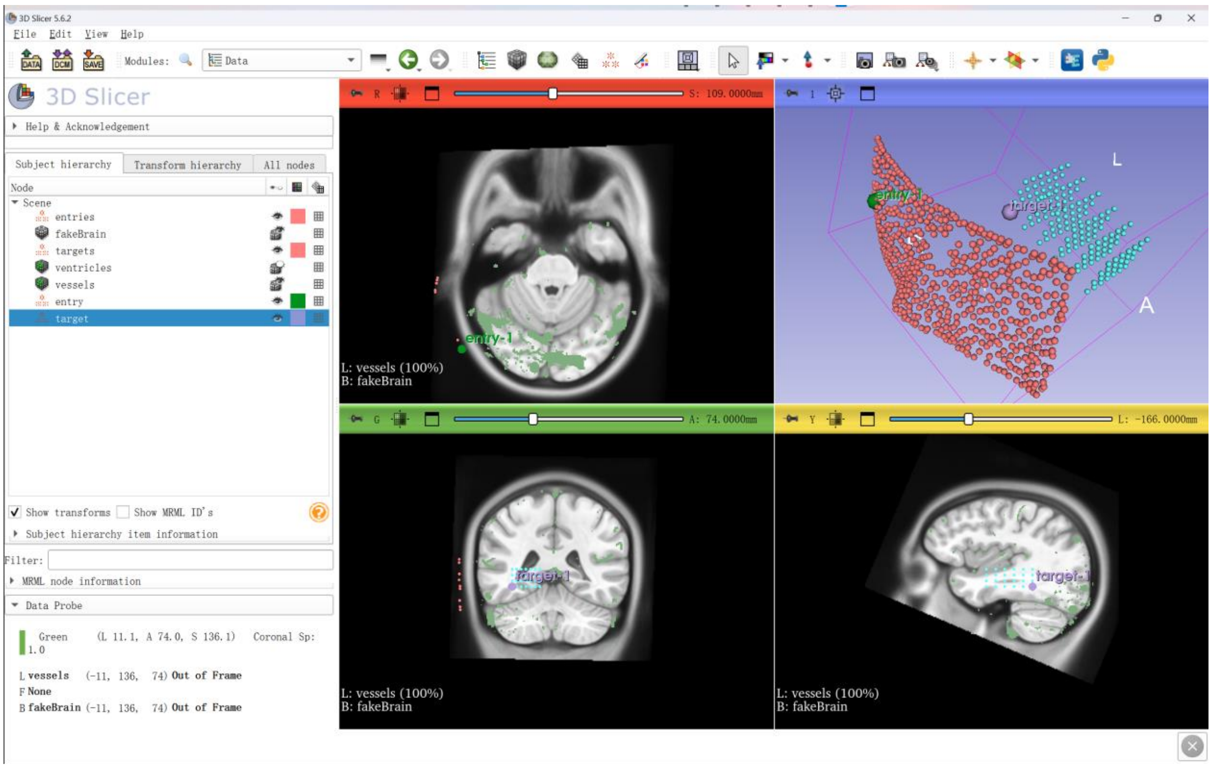


Figure 4: A screenshot containing the entry point (green) and target point(purple) of the best trajectory. The rest color labels are identical to Fig.3

The code adopts a modular design, with each procedural step programmed within its dedicated function. The *findpath()* function acts as an overarching wrapper, enabling users to execute the entire process via a single line of code. The process begins with the *convert_nodes_to_arrays()* function, which accepts a *vtkMRMLLabelMapVolumeNode* (denoting a critical structure) and a *vtkMRMLMarkupsFiducialNode* (representing either an entry or target) as inputs.

While the conversion of the Volume Node to a NumPy array is straightforward through the *slicer.util.arrayFromVolume()* function, accurately mapping the Markups Fiducial Node within the volume to return its coordinates relative to the Volume Node is more complex. This is accomplished by first retrieving the node's location in RAS space using the *.GetNthControlPointPosition()* method, followed by a transformation into IJK coordinates via the *.MultiplyFloatPoint()* method.

The output of this function is an n×3 array representing the Markups Fiducial Node, storing the coordinates of each fiducial point. The Volume Node is returned as a 3-dimensional NumPy array (L×W×H), where the value 0 signifies empty space, and 1 indicates the presence of a blood vessel or ventricle. Later, it will be converted to an n×3 array by *np.argwhere() too*. This function will be called twice to convert all four inputs.

Bresenham's algorithm is efficient for generating a straight line between two points in a coordinate plane. While originally designed for 2D spaces, it can be adapted for 3-dimensional spaces[20,21]. The Python code for this algorithm is readily available online[22]. In this project, I utilized the code mentioned above as the *Bresenham3D()* function, and made slight modifications to integrate it into the *findpath()* data pipeline. This function accepts 2 NumPy coordinate arrays (of shape (3,)) as input and returns an (n,3) array listing the location of every point along the line.

As this algorithm is designed to work with integer coordinates, the rounding of point locations, which are represented as float in 3D Slicer, is necessary. This rounding can lead to a loss of precision and, consequently, reduced accuracy of the algorithm. To mitigate this, the *scale_arrays()* function is employed to expand the NumPy array space—doubling its size by default (with *scale_rate* set to 2)—prior to rounding. While increasing the *scale_rate* can enhance precision, it also elevates the computational complexity by a factor of three. Any duplicates caused by rounding will be removed before the next stage.

The *compute_best_path ()* function is designed to identify the optimal path for a given entry point. This function is tailored to process a single entry at a time for multithreading. Initially, the function computes the Euclidean distance between the entry and target points, disregarding any trajectories that exceed the user-specified maximum length (denoted by the *max_length* argument). Following this, it employs the *Bresenham3D()* function to generate potential paths. The minimum value returned by *distance.cdist()* from the SciPy library is then used to determine the distances between these paths and critical structures. The function returns the path that maintains the longest distance from the critical structures, thereby ensuring the safest route.

Concurrency is employed within the *findpath()* function to enhance computational efficiency. Each entry point is assigned to a separate CPU thread, allowing their optimal trajectories to be calculated in parallel. These best trajectories for each point are stored in a list object (the *paths* variable), and subsequently, the overall best trajectory is determined from it. Finally, the *array2node()* function reverses the changes made by *convert_nodes_to_arrays()*, converting the starting and ending points of this trajectory, which are NumPy arrays, back to format for visualization in 3D Slicer.

2.2 Data Transfer

A *vtkMRMLLinearTransformNode* is created to facilitate the transformation process. Upon executing the *findpath()* function, which retrieves the entry and target points, users can acquire this node through the *gettransform()* method.

After determining the coordinates of the entry and target points and their conversion into NumPy arrays, a vector, or a virtual needle, is created originating from the origin [0, 0, 0] and extending along the x-axis. The magnitude of this vector is defined by the Euclidean distance between the entry and target points, while its orientation always pointing at the positive direction of x-axis (right in 3D Slicer).

The *RotationMatrix()* function is designed to compute the rotation matrix to the target. It accepts three points as inputs: the entry point, the needle tip prior to rotation, and the target location. The function transforms these points into unit vectors and calculates the rotation vector between them. Finally, the *scipy.spatial.transform.Rotation.from_rotvec()*[25] method is invoked to transform the rotation vector into a rotation matrix.

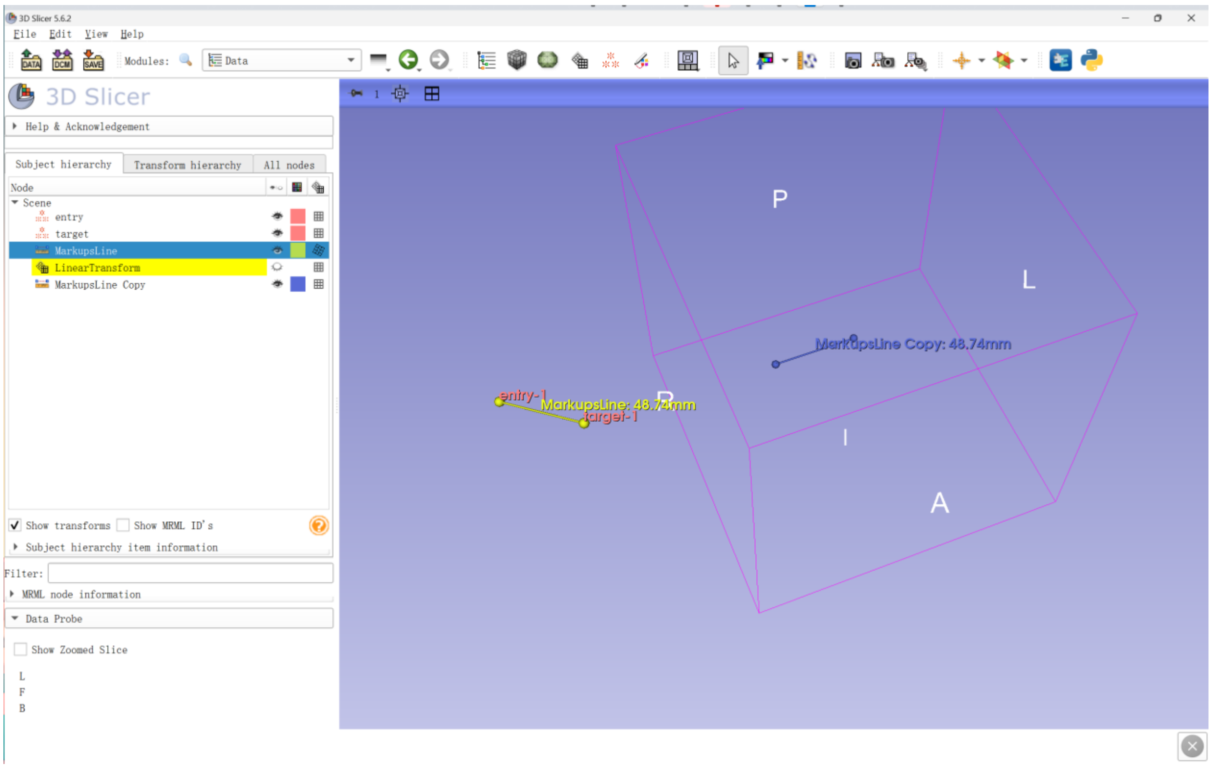


Figure 5: the needle before transformation (blue) and after (yellow)

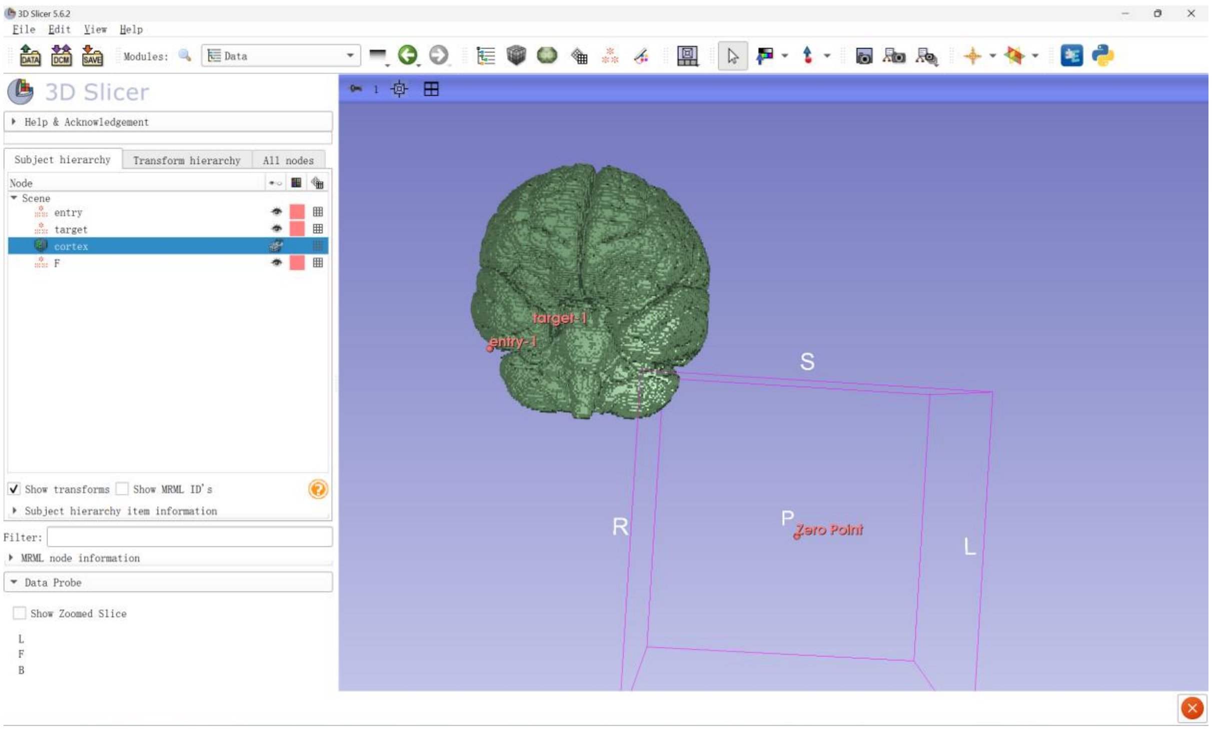


Figure 6: Relevant location of brain and zero point.

In this dataset, the brain is positioned on the right-superior corner of the zero point (Fig. 6). This renders the zero point an invalid starting position, as the end effector should cross the patient and inject from the contralateral side. To rectify this issue, users can provide an additional *patient_position* argument in the *gettransfrom()* function. This will relocate the entry and target points accordingly, simulating an environment where the robot is better positioned (Fig.7).

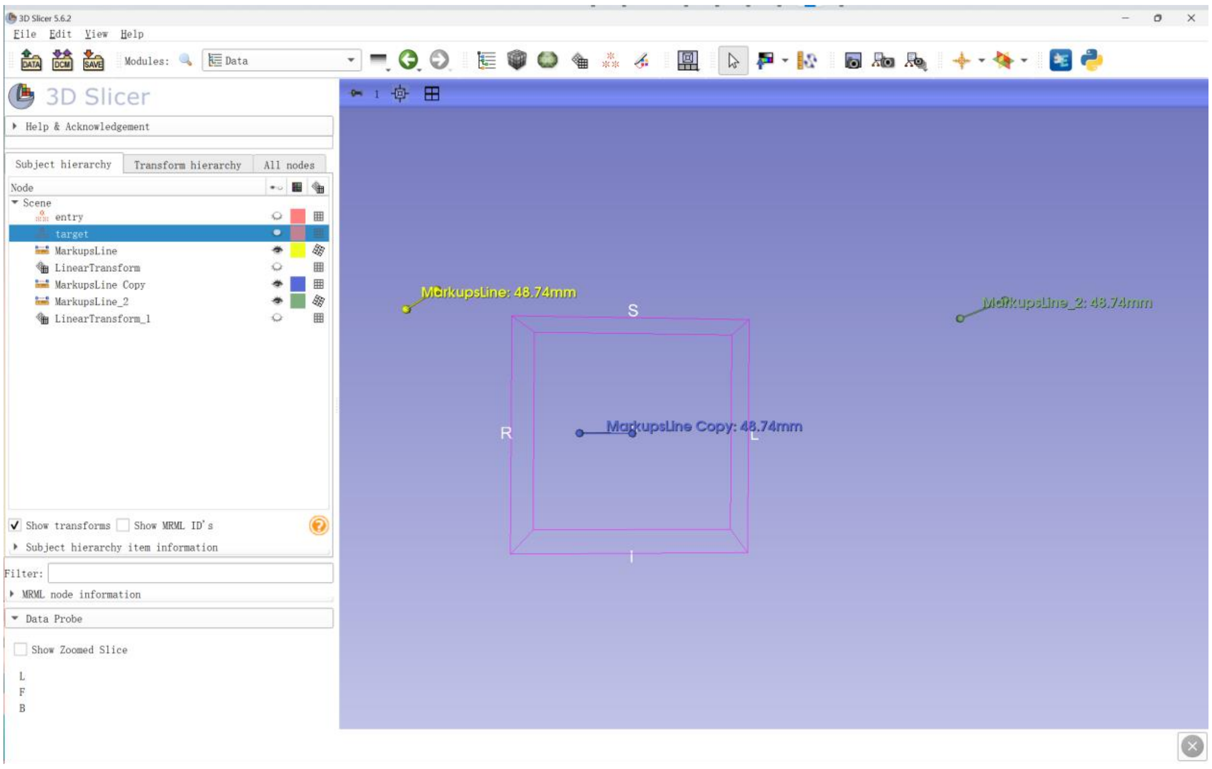


Figure 7: initial position(blue), initial trajectory(yellow), and trajectory from a better placement (green)

2.3 ROS Simulation

This robot is an improved version of the *ImprovedJointedRobot.urdf* file in the week 5 class repository. The manipulators' configuration can be denoted by the sequence: R ↦ R ↦ R ↦ R ↦ R ↦ R ↦ R ↦ R ↦ P. The surgical robot features an arm mounted on a base that connects to the end-effector, which functions as an injector.

In the original file, the pivots' radius was 1, and the joints that connected them to their distal arm were limited to [-180°, 180°]. This limitation on maneuverability has been improved by setting the radius to 2, allowing a wider rotation range. Additionally, the [-180°, 180°] hard constraint has been removed. Now, the rotation range is only limited by the collision detection system in MoveIt.

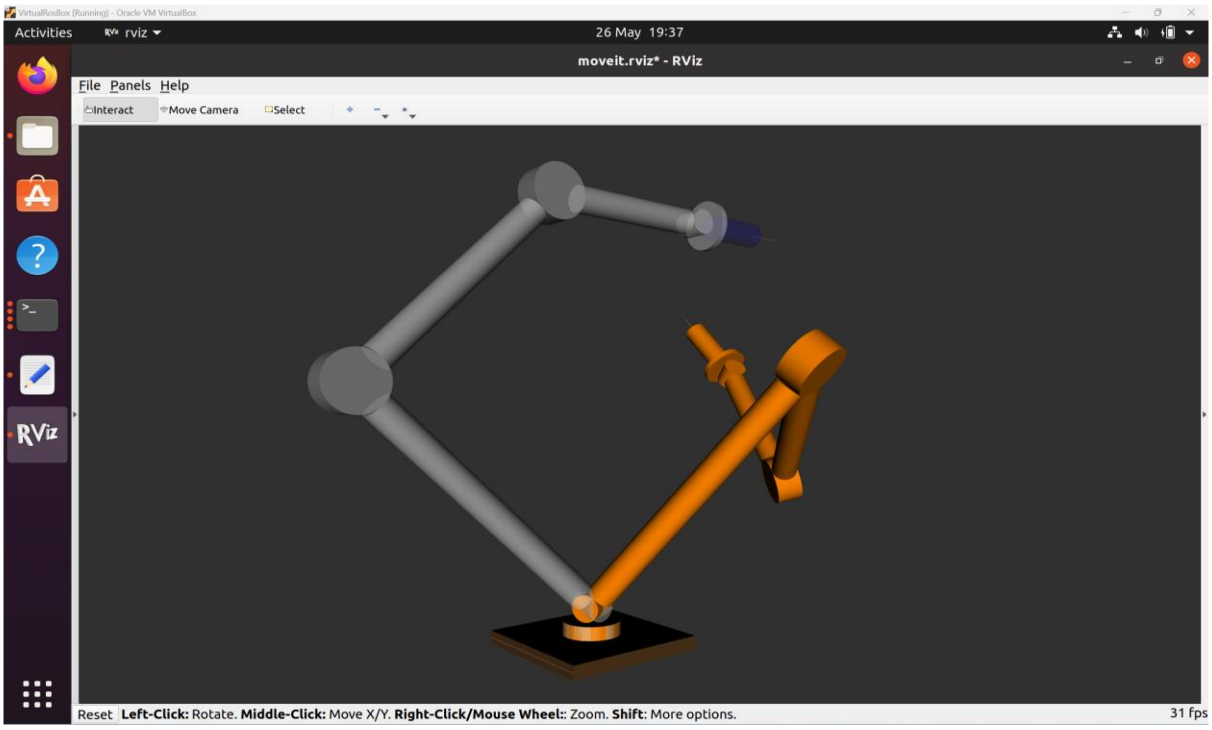

Activities
rviz
26 May 19:37
moveit.rviz* - RViz
File Panels Help
Interact
Move Camera
Select
Reset Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel:: Zoom. Shift: More options.
31 fps

Figure 8: my injection robot in default position (transparent) and random position (orange)

The Move Group Python interface file is located at *MyBot/launch* folder. Prior to initializing everything, the file prompts the user to input two numerical values: the scale rate between the robot and the 3D Slicer model size, and the length of the trajectory. A larger value for the scale rate results in a larger robot relative to the patient's brain. After obtaining these inputs, the file sets up all necessary configurations and moves the robot to a default position, with its end effector pointing directly along the positive x-axis.

Afterward, the program prompts the user to send the *vtkMRMLLinearTransformNode* via OpenIGTLink. The program reads this transformation and converts it from the 3D Slicer coordinate system to ROS. Since both 3D Slicer and ROS2 use a right-handed coordinate system, there is no need for a coordinate system conversion. This alignment has been confirmed during the visual validation phase of the project. The only difference lies in the default view orientation: 3D Slicer's interface is x-left, while ROS2's default view is x-right. However, it can be easily fixed by adjusting your screen view with a rotation.

Note that the transformation matrix within the 3D Slicer space is defined from needle tip to needle tip. Directly planning the movement based on this matrix could result in premature injection, where the needle might penetrate the target surface location before reaching its final pose, or it might not extend at all. This issue is addressed in the *PoseGoal0.2.py* file. Unlike its predecessor *PoseGoal0.1.py*, version 0.2 calculates—without executing—the extension of the joint connecting the needle and syringe. This extension is determined by the ratio of the trajectory length to the scaling rate. With the calculated extension and final orientation (derived from the rotation), the program computes the translation induced by the needle in the form of [x, y, z] coordinates. Subsequently, the arm's movement compensates for the needle insertion. The actual needle insertion is executed as the final step.

3. Validation

3.1 3D Slicer

The path planning algorithm underwent validation through a combination of unit tests and visual inspection. Its efficiency will be discussed in this part as well.

The unit tests comprehensively assessed components where data is stored and processed as NumPy arrays, which are *scale_arrays()*, *Bresenham3D()*, *LineLength()*, *removeduplicates()*, and *RotationMatrix()* functions. However, it's important to note that the unit tests do not encompass *convert_nodes_to_arrays()*, *array2node()*, and *gettransform()*; they involve interactions with the Slicer and VTK libraries, which pose challenges for installation within a Conda environment and independent execution without the 3D Slicer software.

Visual inspection was conducted within 3D Slicer. Although the trajectory does not penetrate the critical structures, it runs tangentially to the inferior cerebral vein, as indicated by the red circle in Figure 9. This error can be attributable to the rounding in *scale_arrays()* function or interpolation process. The critical structures and the trajectory generated by Bresenham's algorithm are point clouds. Nevertheless, in 3D Slicer, these point clouds underwent interpolation and were rendered as continuous lines and solid 3D models. As the result, overlap can occur in the gaps between points.

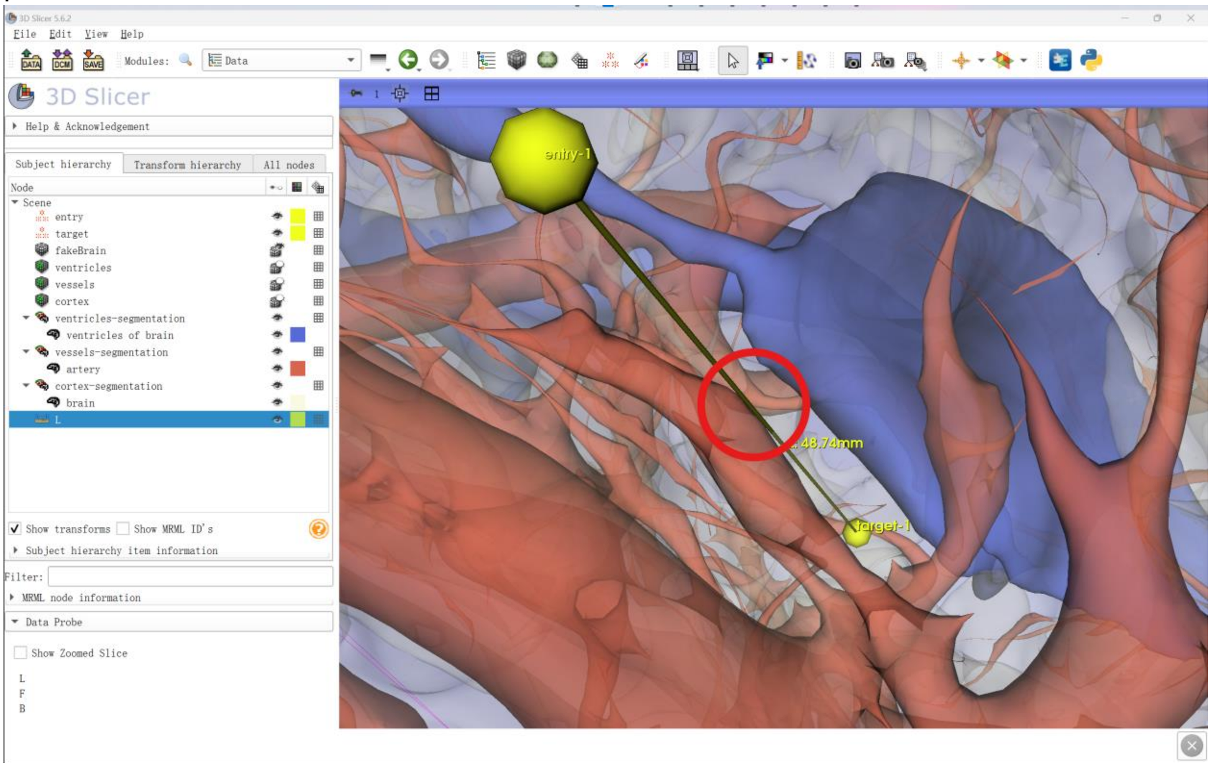
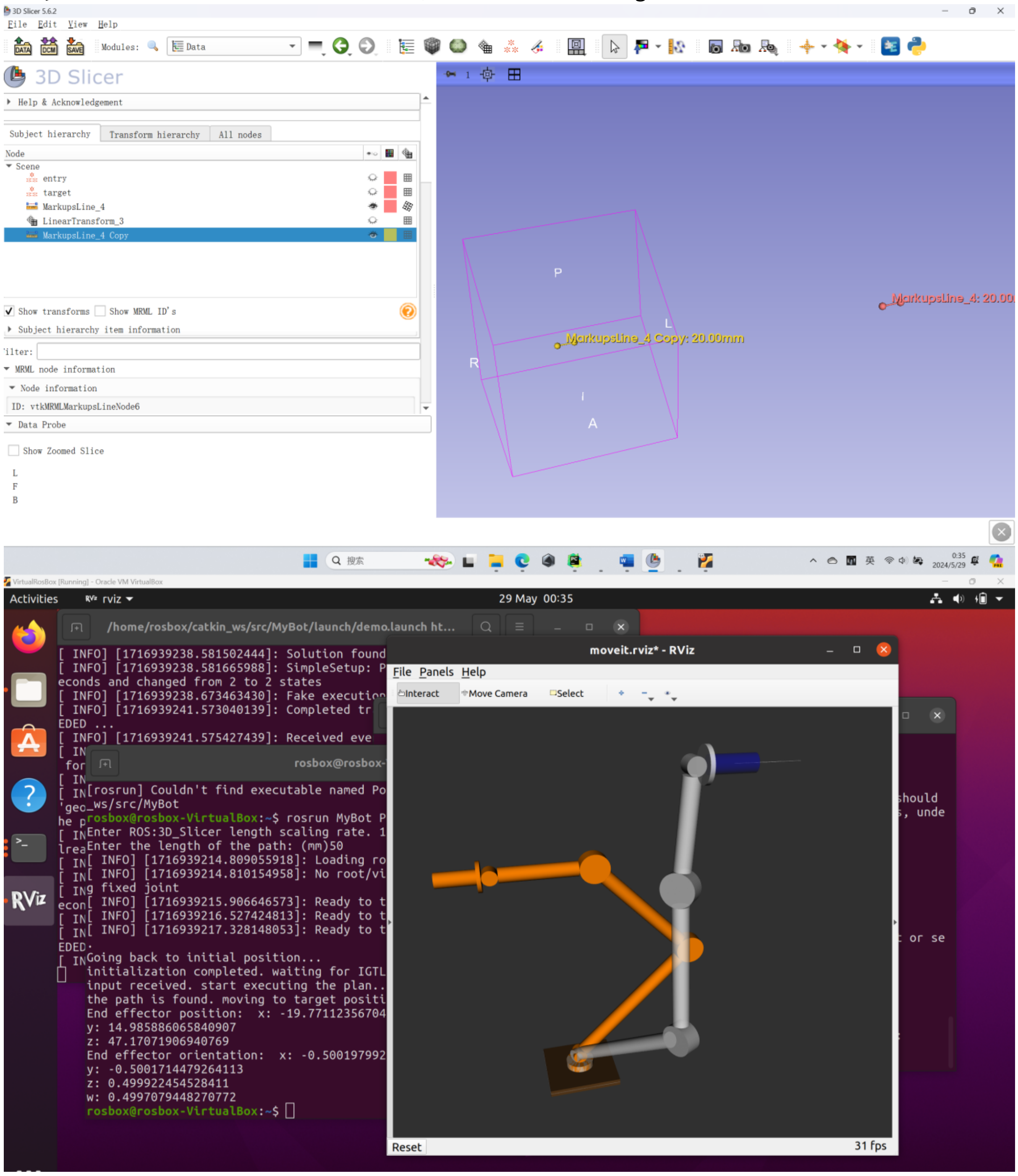


Figure 9: visual inspection of trajectory (yellow) and the tangency (within circle)

NumPy arrays represent structures as points. They are less efficient than meshes for iterative processes, leading to an anticipated slower speed. The time library is integrated into the code to measure time consumption. In my practice, determining the optimal trajectory with *scale_rate=2* for the entire dataset required 154 minutes on my laptop powered by an Intel 13900H CPU and consumed 8GB of RAM.

3.2 ROS2

In ROS, validation was performed through number check and visual inspection. Firstly, generic entry points and targets were utilized to generate *vtkMRMLLinearTransformNode*s to verify proper coordinate conversion. The generic trajectories followed specific axis, which facilitated error identification related to rotation. Proper conversion of translational movements was confirmed visually. As an example, in Fig. 10 the needle is transformed to the superior anterior left corner from its original position, and it undergoes a 180° rotation along the y-axis. This transformation between 3D Slicer and RViz aligns as shown.

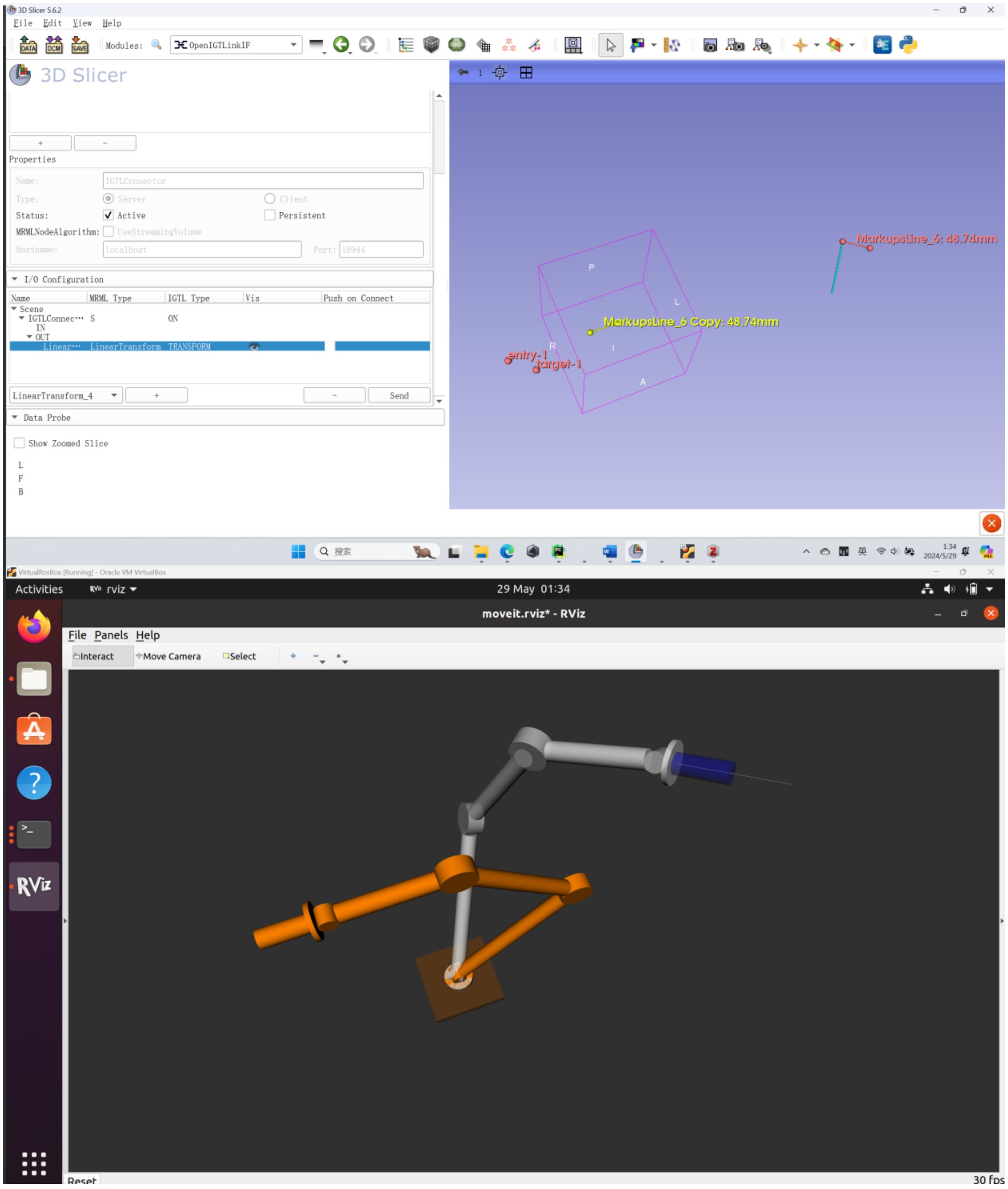


```
/home/rosbox/catkin_ws/src/MyBot/launch/demo.launch ht...
[ INFO] [1716939238.581502444]: Solution found
[ INFO] [1716939238.581665988]: SimpleSetup: P
econds and changed from 2 to 2 states
[ INFO] [1716939238.673463430]: Fake executio
[ INFO] [1716939241.573040139]: Completed tr
EDED ...
[ INFO] [1716939241.575427439]: Received eve
```

```
rosbox@rosbox-
[rosrun] Couldn't find executable named Po
_ws/src/MyBot
rosbox@rosbox-VirtualBox:~$ rosrun MyBot P
Enter ROS:3D_Slicer length scaling rate. 1
Enter the length of the path: (mm)50
[ INFO] [1716939214.809055918]: Loading ro
[ INFO] [1716939214.810154958]: No root/vi
g fixed joint
[ INFO] [1716939215.906646573]: Ready to t
[ INFO] [1716939216.527424813]: Ready to t
[ INFO] [1716939217.328148053]: Ready to t

Going back to initial position...
initialization completed. waiting for IGTL
input received. start executing the plan..
the path is found. moving to target positi
End effector position:  x: -19.77112356704
y: 14.985886065840907
z: 47.17071906940769
End effector orientation:  x: -0.500197992
y: -0.5001714479264113
z: 0.499922454528411
w: 0.4997079448270772
rosbox@rosbox-VirtualBox:~$ 
```

Figure 10: Visual validation of a pair of generic entry and target. Yellow(3D Slicer), orange(RViz): start; red(3D Slicer), colored(RViz): goal



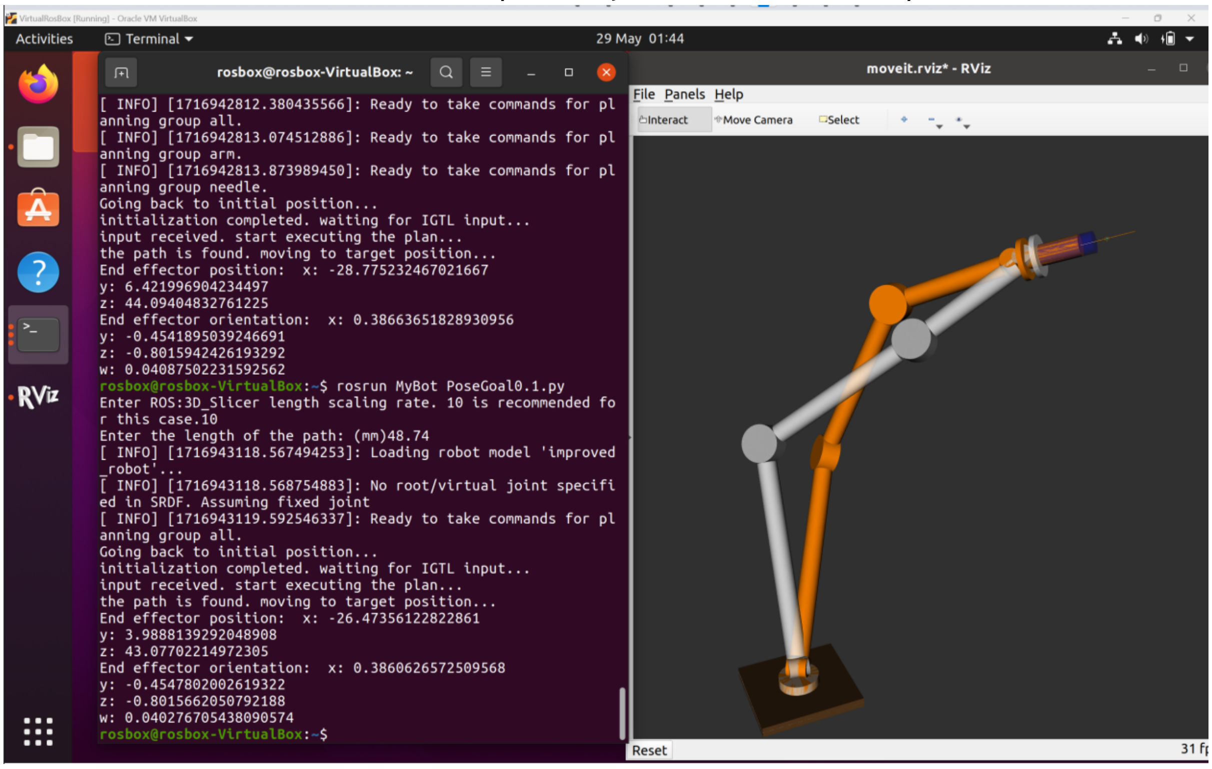
```
Name:               IGTLConnector
Type:               (•) Server        ( ) Client
Status:             [✓] Active        [ ] Persistent
MRMLNodeAlgorithm:  [ ] UseStreamingVolume
Hostname:           localhost          Port: 18944

Name            MRML Type         IGTL Type   Vis   Push on Connect
▾ Scene
  ▾ IGTLConnec…  S                 ON
      IN
    ▾ OUT
        Linear…  LinearTransform   TRANSFORM

LinearTransform_4   +   -   Send
```

Figure 11: Visual Inspection for the optimal trajectory from PathPlanning.py. Color is same as Fig.10.

The same procedure was later applied to the “real” entry and target points from section 2.1, and with different patient placement settings. Additionally, the difference between *PoseGoal0.1.py* and *PoseGoal0.2.py* was validated to ensure that the latter is executed accurately. In Fig. 12, the terminal window displays the end effector position and orientation from v0.2 (upper) as identical to that from v0.1. Furthermore, in RViz, their syringes (end effectors) are aligned, indicating that the translation can be attributed to compensatory movement for needle protraction.



```
[ INFO] [1716942812.380435566]: Ready to take commands for pl
anning group all.
[ INFO] [1716942813.074512886]: Ready to take commands for pl
anning group arm.
[ INFO] [1716942813.873989450]: Ready to take commands for pl
anning group needle.
Going back to initial position...
initialization completed. waiting for IGTL input...
input received. start executing the plan...
the path is found. moving to target position...
End effector position:  x: -28.775232467021667
y: 6.421996904234497
z: 44.09404832761225
End effector orientation:  x: 0.38663651828930956
y: -0.4541895039246691
z: -0.8015942426193292
w: 0.04087502231592562
rosbox@rosbox-VirtualBox:~$ rosrun MyBot PoseGoal0.1.py
Enter ROS:3D_Slicer length scaling rate. 10 is recommended fo
r this case.10
Enter the length of the path: (mm)48.74
[ INFO] [1716943118.567494253]: Loading robot model 'improved
_robot'...
[ INFO] [1716943118.568754883]: No root/virtual joint specifi
ed in SRDF. Assuming fixed joint
[ INFO] [1716943119.592546337]: Ready to take commands for pl
anning group all.
Going back to initial position...
initialization completed. waiting for IGTL input...
input received. start executing the plan...
the path is found. moving to target position...
End effector position:  x: -26.47356122822861
y: 3.9888139292048908
z: 43.07702214972305
End effector orientation:  x: 0.3860626572509568
y: -0.4547802002619322
z: -0.8015662050792188
w: 0.040276705438090574
rosbox@rosbox-VirtualBox:~$
```

Figure 12: Left: Terminal returns same end effector stats. Right: comparison between the end pose from PoseGoal0.1(colored) and PostGoal0.2(orange)

4. Discussion

This project establishes an end-to-end pipeline from medical imaging to surgical path planning. It computes a relatively safe path, accurately transfers this path to ROS, and autonomously plans the robot's movements to reach the targeted position. Nevertheless, there are several areas for improvement.

The primary concern is the performance of the path planning algorithm. As previously mentioned, the point-based pipeline not only compromises efficiency but also introduces inaccuracies due to rounding and interpolation. A potential solution is to transition from using the NumPy and SciPy libraries to more specialized and efficient image processing libraries, such as VTK or ITK.

Other potential future improvements include the following:

1. Brain in RViz Space: Integrating the static anatomical model, or even a digital twin with biophysical properties of a real brain into RViz would allow obstacle avoidance when solving robot kinematic chain. It can also provide better visualization, which can benefit plan making and risk management before surgery.
2. Quantitative Validation Framework: Developing a quantitative validation and evaluation framework for the entire end-to-end pipeline would allow for better assessment.

3. Flexible Path Planning Algorithm: Expanding the path planning algorithm to handle more than two types of obstacles (or fewer) would increase its versatility across different scenarios.

4. User-Friendly Enhancements:

   4.1 GUI: Extending the *PathPlanning.py* file into a 3D Slicer extension with a convenient GUI would simplify user interactions and streamline the workflow.

   4.2 Automated Trajectory Length Input: Enabling the *PoseGoal.py* file to read the trajectory length directly from 3D Slicer would eliminate the need for manual input by users.

References

1. Summers RM, Yao J, Pickhardt PJ, et al. Computed tomographic virtual colonoscopy computer-aided polyp detection in a screening population. Gastroenterology. 2005;129(6):1832-1844. doi:10.1053/j.gastro.2005.08.054

2. Park CC, Brummer ME, Sadigh G, et al. Automated Registration and Color Labeling of Serial 3D Double Inversion Recovery MR Imaging for Detection of Lesion Progression in Multiple Sclerosis. J Digit Imaging. 2023;36(2):450-457. doi:10.1007/s10278-022-00737-1

3. Henson JW, Ulmer S, Harris GJ. Brain Tumor Imaging in Clinical Trials. American Journal of Neuroradiology. 2008;29(3):419-424. doi:10.3174/ajnr.A0963

4. Zhang J, Yang L, Hu Y, et al. Calculation of left ventricular ejection fraction using an 8-layer residual U-Net with deep supervision based on cardiac CT angiography images versus echocardiography: a comparative study. Quant Imaging Med Surg. 2023;13(9):5852-5862. doi:10.21037/qims-22-976

5. Duffy G, Cheng PP, Yuan N, et al. High-Throughput Precision Phenotyping of Left Ventricular Hypertrophy With Cardiovascular Deep Learning. JAMA Cardiol. 2022;7(4):386-395. doi:10.1001/jamacardio.2021.6059

6. Jenkinson MD, Barone DG, Bryant A, et al. Intraoperative imaging technology to maximise extent of resection for glioma. Cochrane Database of Systematic Reviews. 2018;(1). doi:10.1002/14651858.CD012788.pub2

7. Bohl DD, Nolte MT, Ong K, Lau E, Calkins TE, Della Valle CJ. Computer-Assisted Navigation Is Associated with Reductions in the Rates of Dislocation and Acetabular Component Revision Following Primary Total Hip Arthroplasty. JBJS. 2019;101(3):250. doi:10.2106/JBJS.18.00108

8. Agarwal S, Eckhard L, Walter WL, et al. The Use of Computer Navigation in Total Hip Arthroplasty Is Associated with a Reduced Rate of Revision for Dislocation: A Study of 6,912 Navigated THA Procedures from the Australian Orthopaedic Association National Joint Replacement Registry. J Bone Joint Surg Am. 2021;103(20):1900-1905. doi:10.2106/JBJS.20.00950

9. Agarwal S, Eckhard L, Walter WL, et al. The Use of Computer Navigation in Total Hip Arthroplasty Is Associated with a Reduced Rate of Revision for Dislocation: A Study of 6,912 Navigated THA Procedures from the Australian Orthopaedic Association National Joint

Replacement Registry. J Bone Joint Surg Am. 2021;103(20):1900-1905. doi:10.2106/JBJS.20.00950

10. Ferretti G, Knoplioch J, Coulomb M, Brambilla C, Cinquin P. [Endoluminal 3D reconstruction of the tracheo-bronchial tree (virtual bronchoscopy)]. J Radiol. 1995;76(8):531-534.

11. Hoppe H, Dinkel HP, Walder B, Allmen G von, Gugger M, Vock P. Grading Airway Stenosis Down to the Segmental Level Using Virtual Bronchoscopy. CHEST. 2004;125(2):704-711. doi:10.1378/chest.125.2.704

12. Asano F, Eberhardt R, Herth FJF. Virtual bronchoscopic navigation for peripheral pulmonary lesions. Respiration. 2014;88(5):430-440. doi:10.1159/000367900

13. Wong KY, Tse H nam, Pak KKT, et al. Integrated use of virtual bronchoscopy and endobronchial ultrasonography on the diagnosis of peripheral lung lesions. J Bronchology Interv Pulmonol. 2014;21(1):14-20. doi:10.1097/LBR.0000000000000027

14. Pesapane F, Rotili A, Penco S, Nicosia L, Cassano E. Digital Twins in Radiology. J Clin Med. 2022;11(21):6553. doi:10.3390/jcm11216553

15. Herron DM, Marohn M, The SAGES-MIRA Robotic Surgery Consensus Group. A consensus document on robotic surgery. Surg Endosc. 2008;22(2):313-325. doi:10.1007/s00464-007-9727-5

16. Tao F, Cheng J, Qi Q, Zhang M, Zhang H, Sui F. Digital twin-driven product design, manufacturing and service with big data. Int J Adv Manuf Technol. 2018;94(9):3563-3576. doi:10.1007/s00170-017-0233-1

17. Modules:OpenIGTLinkIF-Documentation-3.4 - Slicer Wiki. Accessed May 20, 2024. https://www.slicer.org/wiki/Modules:OpenIGTLinkIF-Documentation-3.4

18. Building a visual robot model from scratch — ROS 2 Documentation: Humble documentation. Accessed May 21, 2024. https://docs.ros.org/en/humble/Tutorials/Intermediate/URDF/Building-a-Visual-Robot-Model-with-URDF-from-Scratch.html

19. MoveIt Setup Assistant — moveit_tutorials Kinetic documentation. Accessed May 20, 2024. https://docs.ros.org/en/kinetic/api/moveit_tutorials/html/doc/setup_assistant/setup_assistant_tutorial.html

20. Au C, Woo T. Three dimensional extension of Bresenham's Algorithm with Voronoi diagram. Computer-Aided Design. 2011;43(4):417-426. doi:10.1016/j.cad.2010.11.006

21. Liu XW, Cheng K. Three-dimensional extension of Bresenham's algorithm and its application in straight-line interpolation. Proceedings of the Institution of Mechanical Engineers, Part B: Journal of Engineering Manufacture. 2002;216(3):459-463. doi:10.1243/0954405021519979

22. Bresenham's Algorithm for 3-D Line Drawing. GeeksforGeeks. Published July 15, 2018. Accessed May 20, 2024. https://www.geeksforgeeks.org/bresenhams-algorithm-for-3-d-line-drawing/

23. urdf/XML/joint - ROS Wiki. Accessed May 21, 2024. https://wiki.ros.org/urdf/XML/joint

24. Move Group Python Interface — moveit_tutorials Melodic documentation. Accessed May 26, 2024.

https://docs.ros.org/en/melodic/api/moveit_tutorials/html/doc/move_group_python_interface/move_group_python_interface_tutorial.html

25. scipy.spatial.transform.Rotation.from_rotvec — SciPy v1.13.1 Manual. Accessed May 29, 2024. https://docs.scipy.org/doc/scipy/reference/generated/scipy.spatial.transform.Rotation.from_rotvec.html