Spinal Navigation Project

ESE 499 Capstone Design Project Final Report
Submitted to Professor Wang and the Department of Electrical and Systems Engineering
December 15, 2023

Client:
Electrical & Systems Engineering Department

Advisor:
Dorothy Wang[1] – dorothyw@wustl.edu

Collaborator:
Nicholas Pallotta[2] – npallotta@wustl.edu

Engineers:
Seif Elkhashab[3] – e.seif@wustl.edu
Xiangping Ouyang[4] – o.xiangping@wustl.edu
Mark Charnot[5] – c.mark@wustl.edu

Washington University in St. Louis
McKelvey School of Engineering

[1] Lecture, Department of Electrical and System Engineering
[2] Assistant Professor, Department of Orthopedic Surgery
[3] Candidate for B.S. in Systems Science and Engineering
[4] Candidate for B.S. in Computer Engineering
[5] Candidate for B.S. in Systems Science and Engineering

Abstract:

Under the advisory of Dr. Dorothy Wang, this capstone will develop a spinal navigation system, which can model a real spine in virtual environment. Using this model, orthopedic surgeons can assess a patient's misaligned bones during the corrective procedure, rather than waiting for post-surgery CT scan results. The surgeon can make alignments using the virtual model, where attributes like desired length, relative angle, and rotation can be compared to the patient's real-time measurements. This tool gives surgeons the ability to complete the procedure in a more efficient manner, eliminating the time and cost of using CT Scans to determine if the surgery goals have been met. Dr. Nicholas Pallotta from the Department of Orthopedic Surgery at Washington University in St. Louis is providing all the necessary equipment required for this development, including patient CT scans, the NDI Cygna-6D camera, three-dimensional markers for the camera to register, and a life-like spine model. The camera will be coded to detect a minimum of two markers that are drilled or clipped onto the spine. From here, these points will be registered in 3D space, where linear operators will be used to determine the translational and rotational distances between them. Finally, a pre-surgery CT scan will be used as an input, such that image processing can be used to create an exact relation between the vertebrae and the three-dimensional markers.

Introduction:

Background Information:

Spinal deformity occurs when there is an abnormal alignment in the spine [1]. Patients with this condition will feel off balance and can suffer a significant amount of pain [2]. To solve this issue, it's common and essential for patients to undergo spinal deformity correction surgery. During this surgery, the most important step is to obtain the correct shape of the spine and to bend the shape into place manually. Previously, the only way to measure the spine reliably was to obtain CT images after the deformity correction has been performed. Then the doctors will compare these images and preoperative films during the operation. Producing these images and films takes costly in terms of time and money. With that, the doctors also spend significant time reviewing the results and taking measurements from these images. If the alignment does not match the goal for surgery, then all the procedures for spine alignment need to be redone until it reaches a more appropriate position. These will add a huge amount of time to the already lengthy surgery procedure.

Recently, numerous new technologies have been adopted in the field of spine surgery. Luckily, one of the new technologies, called spinal navigation, can allow real-time measurements so that the spine position can be provided during the deformity correction, minimizing the use of CT scans to determine if the surgery's goal have been met. Spinal navigation uses 3 dimensional trackers and cameras to measure the relative positions of the spine to different tools that doctors use to implants they place in the spine. Currently, there is a lack of measurement of the spine intraoperatively for spinal navigation. If this technique is successfully adopted, it will save a tremendous amount of effect and time for spine surgery.

Problem Statement:

The goal of this project is to bridge the gap between the existing hardware and the orthopedic surgeon's goal, which is developing a methodology to observe the bone's rotation and translation in real-time. Currently, the hardware only measures the translation and rotation of the trackers. However, these measurements are not useful, as the medical team had strict goals for the corrective procedure, so we must have precise vertebrae measurements. Since the camera does not have any knowledge of the bone's movement, we must create a linear relationship between the tracker and bone's movement.

Objectives of the Project:

In order to create a relationship between the 3D trackers and the bones, there are several objectives that must be met. These objectives are listed below:

1. Create a program to classify the 3D trackers within a CT scan.
2. Create a program to classify bones within a CT scan.
3. Create a program that uses the bone and tracker locations as input, and returns the linear transformation needed to relate the trackers to the bone.

Each of these methods is explored thoroughly in the 'Methods' section.

Required Hardware and Data

Designing this solution required a variety of data and hardware. First, to create a transformation between the trackers and the bones of interest, our team needed to use some data format that would allow this to be possible. Because a pre-surgery CT scan is already a standard process, we found it useful to ask for a CT scan that includes both the bones and the trackers. From here, we could iterate through the slices of the scan to find the bones and trackers.

The first piece of data needed was a CT scan of the desired operation area. More specifically, it is crucial that this CT scan was from an operation where these 3D trackers are in place. Without these in place, it would be impossible to create a registration between the trackers and the bones of interest. This data type was limited, as this tool has never been used in a real application. To bypass this issue, our team requested a CT scan of a model spine with the trackers in place.

Since it was also important to accurately register the bone in the CT scans, our team found it useful to acquire additional CT scans, even though they do not have the 3D markers in place. This data can be used to test our bone registration algorithm's robustness. Although these were not useful towards creating a relationship between the bone and the tracker, the abundance of this data type was critical for refining the registration algorithm. One frame of such a CT scan is displayed below in Figure 1. Like the other CT scans, these were provided by Dr. Pallotta.

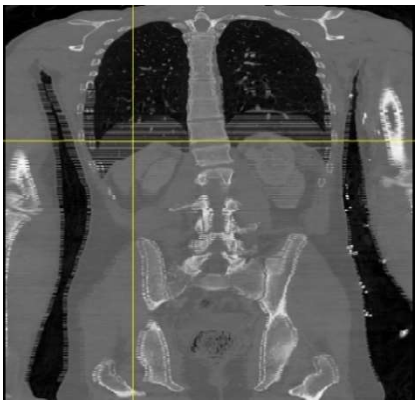


Figure 1: Example of CT scan frame

The NDI Cygna-6D camera, along with the 3D trackers are the devices that are responsible for feeding position and rotation data into our program, such that it can be transformed to model the bones. These two pieces of hardware are displayed below in Figure 2.

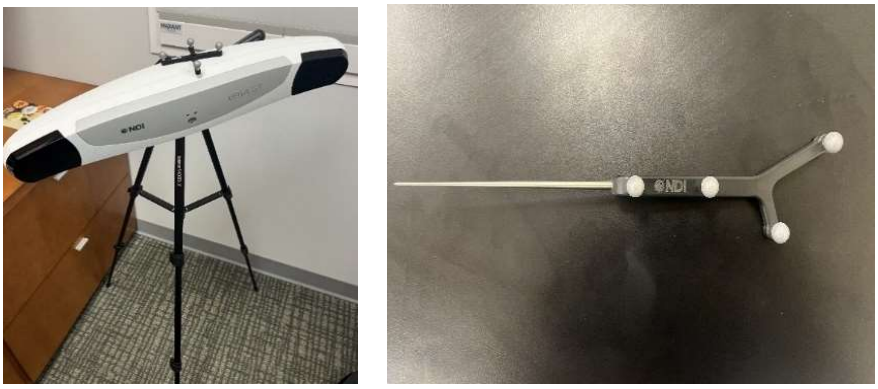


Figure 2: NDI Camera (left) and Tracker (right)

These pieces of hardware, in union with the prepackaged software, output data in the format $v \in \mathbb{R}^6$, where each element of v is a positional or rotation measurement. An additional vector is included in the output for every tracker in the frame of the camera. This data type was easily collected by our team, as we had all the hardware and software needed for collection.

Methods

The following sections highlight the main algorithms involved in the completion of the project deliverables.

Tracker Classification Algorithm (TCA):

The goal of the TCA is to robustly and accurately classify the trackers' locations from a single CT scan provided at the beginning of the surgery. Once the user takes a CT scan of the patient, that will be the only input needed for the TCA to compute the trackers' positions as well as the orientation of the tracker.

TCA Input:

- A folder full of CT scans in the DICOM file format.

- The folder usually contains between 150-300 slices although the number of slices does not affect the performance of the system.

Uploading and transforming the input:

1. The python script starts by importing all the necessary libraries.
2. The user needs to specify the location/path of the folder containing the DICOM files in the python script.
3. The script then automatically recognizes how many files are in the folder and converts all the DICOM files into PNG files as can be seen in Figure 3 and places them into a new folder with a name specified by the user. This is done as PNG files are easier to use with moder image processing libraries and easier to manipulate directly, the pixel relationship is maintained in this relationship.

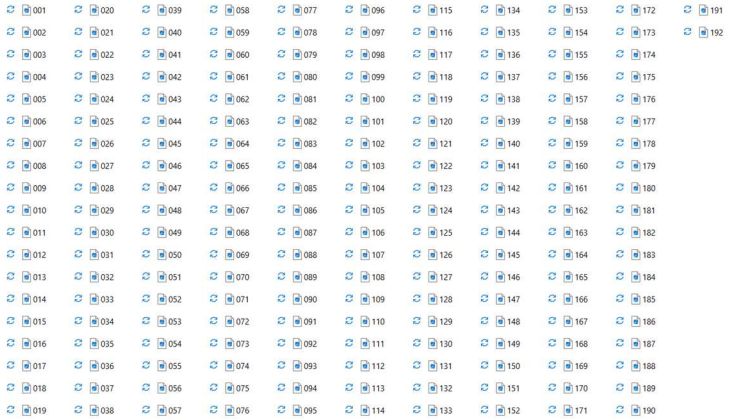


Figure 3: The created folder with all the converted PNG files.

4. While the DICOM files are being transformed, a function will also read one of the DICOM files and print out important parameters of the CT scan such as pixel spacing and slice thickness. This will be done simultaneously with step 3 and is shown in Figure 4.

```
DICOM Parameters (Pixel Spacing, Slice Thickness): ([0.415, 0.415], '0.833')
```

Figure 4: Shows the outputted slice thickness and pixel spacing.

Pre-processing the images:

1. Now that there is a folder of PNGs, the images need to be processed to make the classification algorithm work more accurately and more robustly. Figure 5 shows the image before any preprocessing and after the processing. The image was brightened, converted to grayscale, made binary, and contours will be filled in.

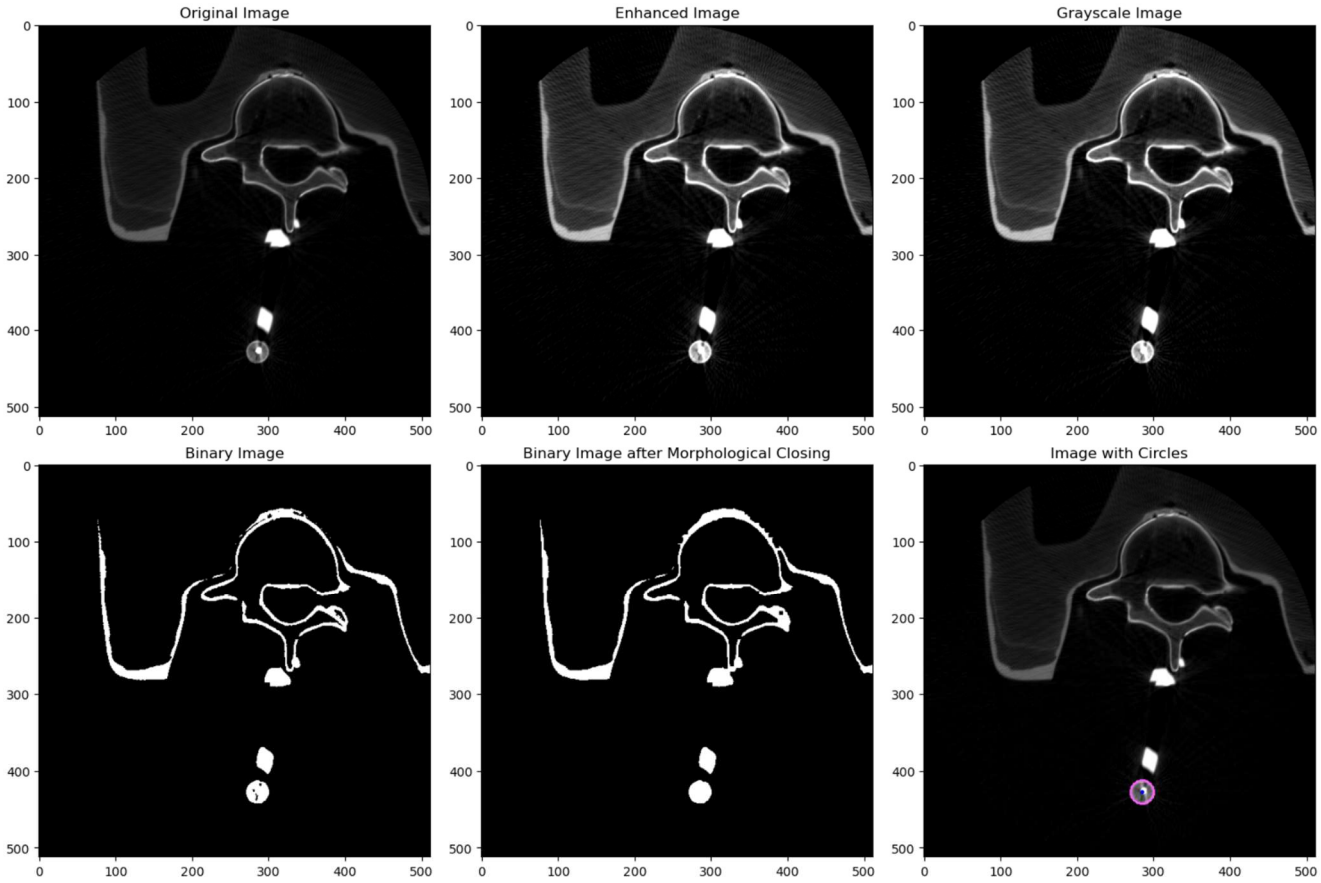


Figure 5: The Preprocessing Steps

Classifying the Trackers (circles):

1. Now that the preprocessing is done and the image is much clearer and more continuous, the circles can begin to be classified.
2. A circularity parameter and a circularity threshold are defined, with the threshold being a design choice. The threshold value was found through a trial and error to find the threshold that most accurately and precisely classifies the trackers, the optimal circularity threshold was found to be 0.8.
3. Further limits were put on what can be classified as a circle by limiting the area of the circle between 100 and 2000 pixels, which aligns with the physical size of the tracker balls.
4. The script has the option to display all the images with the trackers outlined in pink as shown in Figure 6 and prints out/saves the pixel coordinates of the classified trackers as well as their circularity parameter. This also helped in deciding the circularity threshold since it can easily be seen in the output.

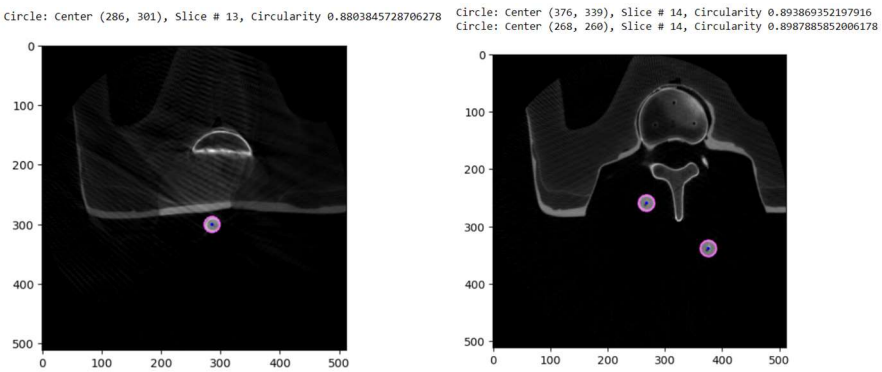


Figure 6: Shows the classified circles in the CT slices.

5. It is important to note that multiple trackers can be seen in the same slice which was a problem early on in the project but by using the circularity method, it's a matter of the circularity of the circles and not the number of circles so this proved to be a very robust method of recognizing all the circles in a slice no matter how many circles are present as seen in Figure 6**.**

Clustering the Tracker Coordinates:

1. Since all the trackers all the trackers are now classified, the algorithm can be made even more robust by ensuring that a ball appears in multiple scans in a row.
2. A circle can be ensured to be a tracker and not an outlier by appearing around the same (x, y) coordinate across adjacent scans. A threshold of 6 continuous scans was set for a circle to be considered a tracker and the set of continuous scans was collected and saved in the PNGS folder from earlier, each cluster of images was saved in its own folder.

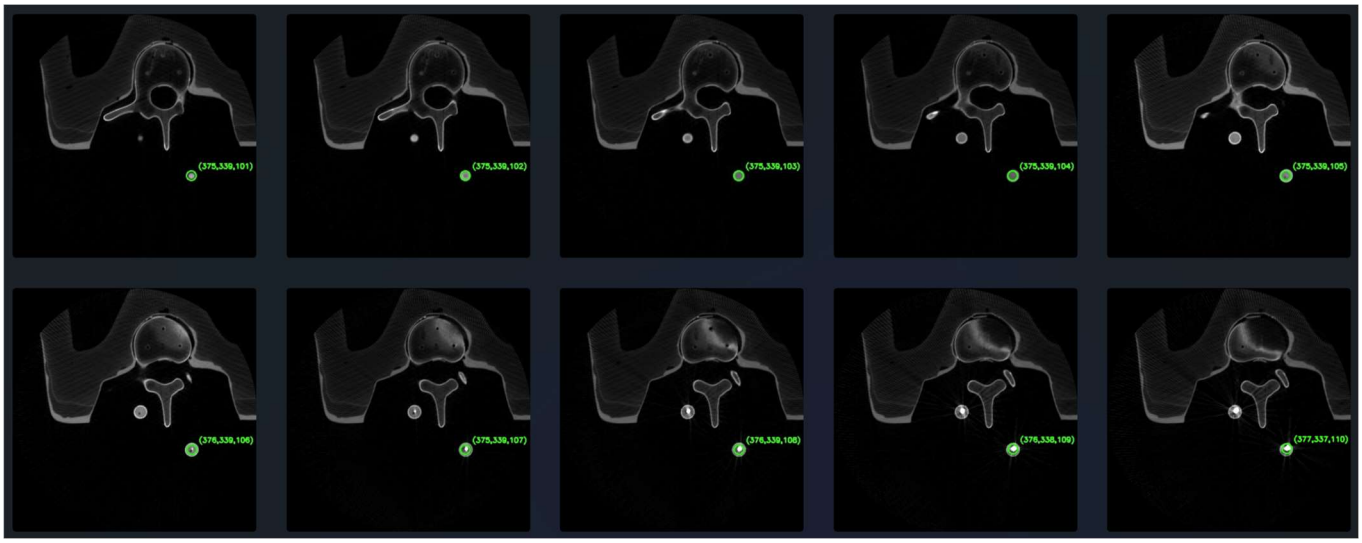


Figure 7: Shows one of the created clusters.

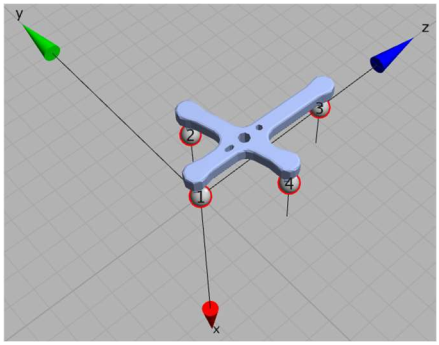
3. Figure 7 shows one of the created clusters and it can be seen how this method ensures with nearly 100% certainty that the trackers are correctly classified. Furthermore, if a scan has more than one tracker, a separate cluster will be made for each tracker.
4. Now that the clusters are made, an average of the median 3 slices will give the most precise coordinate for the center of that respective tracker and the output can be simplified to what is shown in Figure 8.

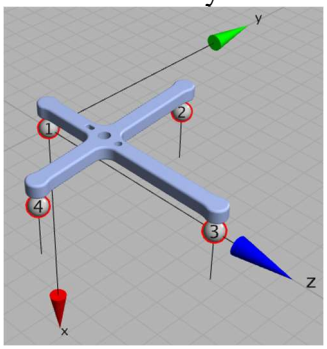
```
Ball 0: Average Coordinates: (286.0, 301.0, 3.5)
Ball 1: Average Coordinates: (268.0, 260.0, 105.5)
Ball 2: Average Coordinates: (374.0, 375.0, 22.5)
Ball 3: Average Coordinates: (203.0, 396.5, 37.5)
Ball 5: Average Coordinates: (332.0, 479.0, 55.5)
Ball 8: Average Coordinates: (375.5, 339.0, 105.5)
Ball 9: Average Coordinates: (198.0, 308.0, 147.5)
Ball 11: Average Coordinates: (285.0, 428.0, 171.5)
```

Figure 8: Shows the final coordinate of the trackers.

Defining Tracker Object, Name & Shape:

1. Since different tracker shapes may be used in a surgery, each tracker shape must be known and defined in the script.
2. A tracker object is defined with an automatically generated name, a user inputted tracker shape chosen from a displayed list, and the 4 ball locations are initialized at the origin, these coordinates will be updated in the next step.
3. This is an important step as each ball must be classified as ball 1, ball 2, ball 3, or ball 4, as this knowledge will give us the information necessary to understand the orientation of the tracker in relation to its coordinate frame set by the camera.






```
# Define the known tracker shapes with ball labels
tracker_shapes = {
    "cross_shape_1": [
        {"ball1": (0.00, 0.00, 0.00)},
        {"ball2": (0.00, 28.59, 41.02)},
        {"ball3": (0.00, 0.00, 88.00)},
        {"ball4": (0.00, -44.32, 40.45)}
    ],
    "cross_shape_2": [
        {"ball1": (0.00, 0.00, 0.00)},
        {"ball2": (0.00, 47.38, 28.99)},
        {"ball3": (0.00, 0.00, 89.10)},
        {"ball4": (0.00, -35.36, 35.36)}
    ]
}
```

Figure 9: The coordinate frame of two of the trackers, tracker data in the script

4. As can be seen in Figure 9**,** there are different tracker shapes, that will need to be inputted into the script so it can successfully classify and differentiate the individual balls into their respective trackers, this is as simple for the user as inputting 1 or 2 into a text box.

```
Tracker Name: Tracker_1, Type: cross_shape_1, Ball Locations: [(0, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0)]
Tracker Name: Tracker_2, Type: cross_shape_2, Ball Locations: [(0, 0, 0), (0, 0, 0), (0, 0, 0), (0, 0, 0)]
```

Figure 10: Shows two tracker objects created with two different shapes.

5. As more tracker shapes are added they can be easily added to the script and the tracker data can be found in the NDI Cygna Camera Software. Figure 9 shows what this looks like in the script for future additions, and Figure 10 shows the output with the initialized coordinates.

Grouping the Clusters into Trackers:

1. All tracker balls are now classified and ready to be grouped into 4's since all trackers have 4 balls on it.
2. The trackers were grouped by knowing that all 4 balls for all trackers lie on a single plane, a script was created to calculate how close to a plane (planarity) a set of 4 balls is, and top sets of balls that create the highest planarity were chosen and grouped into trackers.

```
Plane: ((268.0, 260.0, 105.5), (375.5, 339.0, 105.5), (198.0, 308.0, 147.5), (285.0, 428.0, 171.5)), Planarity: 0.28385832072617595
Plane: ((286.0, 301.0, 3.5), (374.0, 375.0, 22.5), (203.0, 396.5, 37.5), (332.0, 479.0, 55.5)), Planarity: 1.6131346558250432
```

Figure 11: Output of the coordinates found to lie on the same plane.

3. Figure 11¸ Shows the top two planes found by the algorithm and thus the balls were successfully segmented into the correct tracker, the lower the planarity parameter the better and the script has functionality to print out coordinate combinations, most of the other planarites were orders of magnitude higher which supports the robustness of this method.

Classifying each ball to its number:

1. Because of the know relationship of each ball for a given tracker shape, the ball number can be found using a triangulation algorithm. The algorithm used the distance between the balls as well as the angles to determine each ball's number. The completed TCA with each ball classified and belonging to a tracker can be seen below in Figure 12.

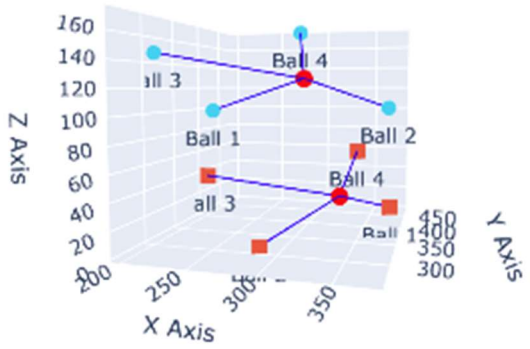


Figure 12: 3D plot of the two trackers from the CT scan.

Bone Classification Algorithm (BCA):

The goal of the BCA is to return lines that model edges of select vertebrae. Like the TCA, the input is a CT scan that is taken before surgery begins. Additionally, this algorithm requires some user input, as the orthopedic surgeon must visually inspect the CT scan and determine which bones, he/she would like track. This is a feature that was requested by Dr. Pallotta, as surgeons often find themselves targeting different objectives, and no surgery is the same.

During development, our team took two approaches to create these line models. First, we attempted to classify the bones using a machine learning method. This approach had potential to be very robust, however, due to the limited time on the project, our team was unable to complete this segment. This could be a great improvement for the next iteration of this project. Our second approach, although more simplistic, was reliably able to model the desired bones as line segments. Detailed descriptions of these methods are listed below.

Machine Learning BCA

Machine learning is a powerful tool for performing biomedical image segmentation tasks, such as segmentation and classification. Among them, convolutional neural networks have been tested for bone classification algorithms on various datasets. Those models include U-Net [2-3], U-Net 3D [4], and V-NET [5]. We adopted the 3D V-Net model to segment the bone position because V-Net exploits down-convolutions with customized stride step and kernel-size, instead of a fixed down sampling size achieved through max-pooling. Learnable parameters are also added in this stage. The network architecture is shown in Figure 13.

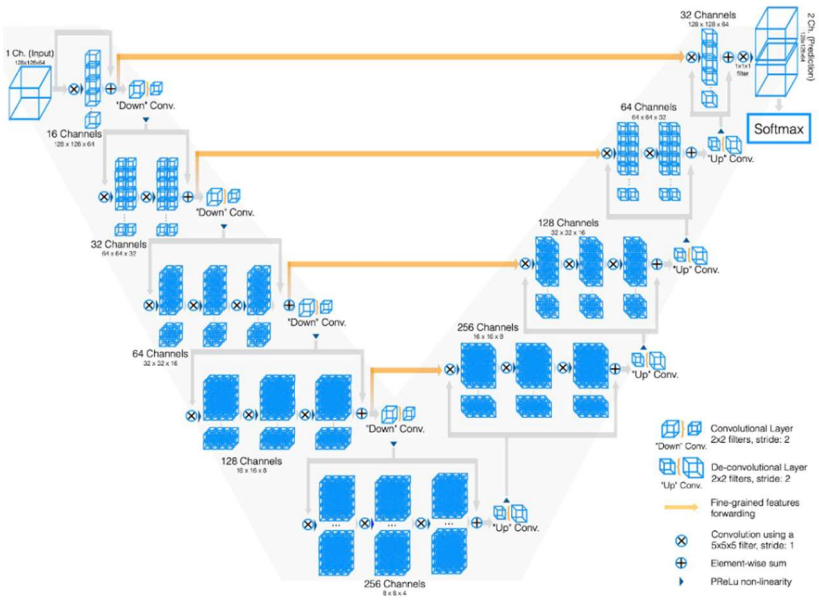


Figure 13: V-Net Architecture.

The input and the output of the model are 3D nii.gz files, which need the type conversion from the original CT scan. This can be achieved through the "dicom2nifti" function in the "convert2nii.py." We chose the Verse data set as the training dataset, including 143 pairs of training and 256 pairs of testing datasets. During the training process, there were lots of difficulties and bugs during the training process, but we finally figured out most of them and got an acceptable test result. It takes roughly two days to train the model with 300 epochs and 143 pairs of training data. Theoretically speaking, the model is supposed to take the original CT scan and crop the image to get rid of the background as much as possible but keep all the bone parts, as shown under the pre-processing in Figure 14. Next, we test the image and get the prediction mask shown in red under the Binary Segmentation image.

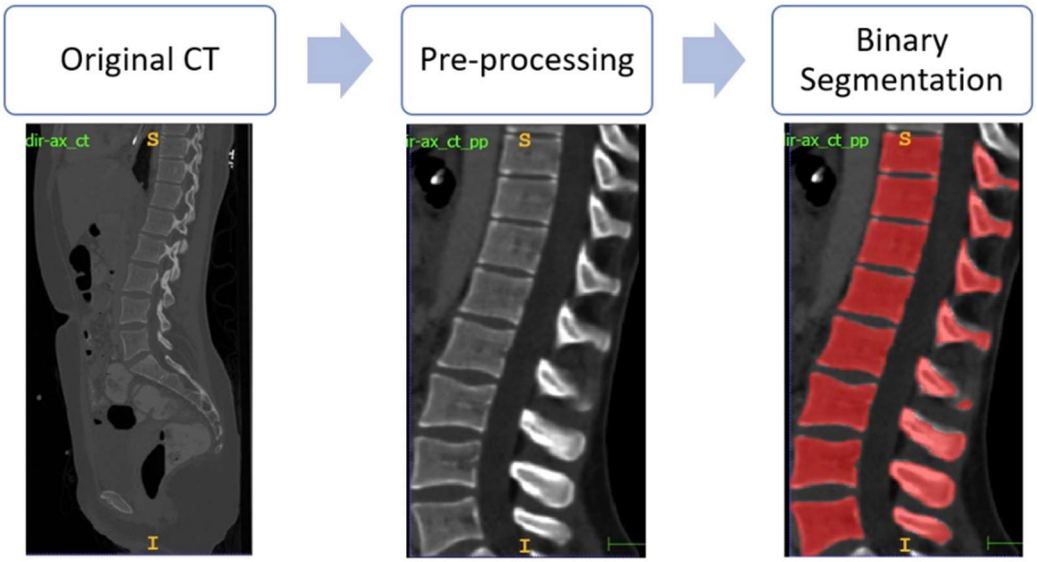


Figure 14: Theoretical results example, desired results.

We also tried other networks, like 3D U-Net as can be seen in Figure 15. U-Net is the most common convolutional neural network in biomedical image segmentation. 3D U-Net maintains the advantages U-Net has and performs 3D operations to keep the 3D information. In our 3D U-Net model, all the images were cropped to a fixed size of 256x256x256. This size is larger than the original image so that all the information from the input images is maintained. Next, the cropped images were fed into the network to train the model. Unfortunately, unsolvable type errors occur that stop us from continuing.

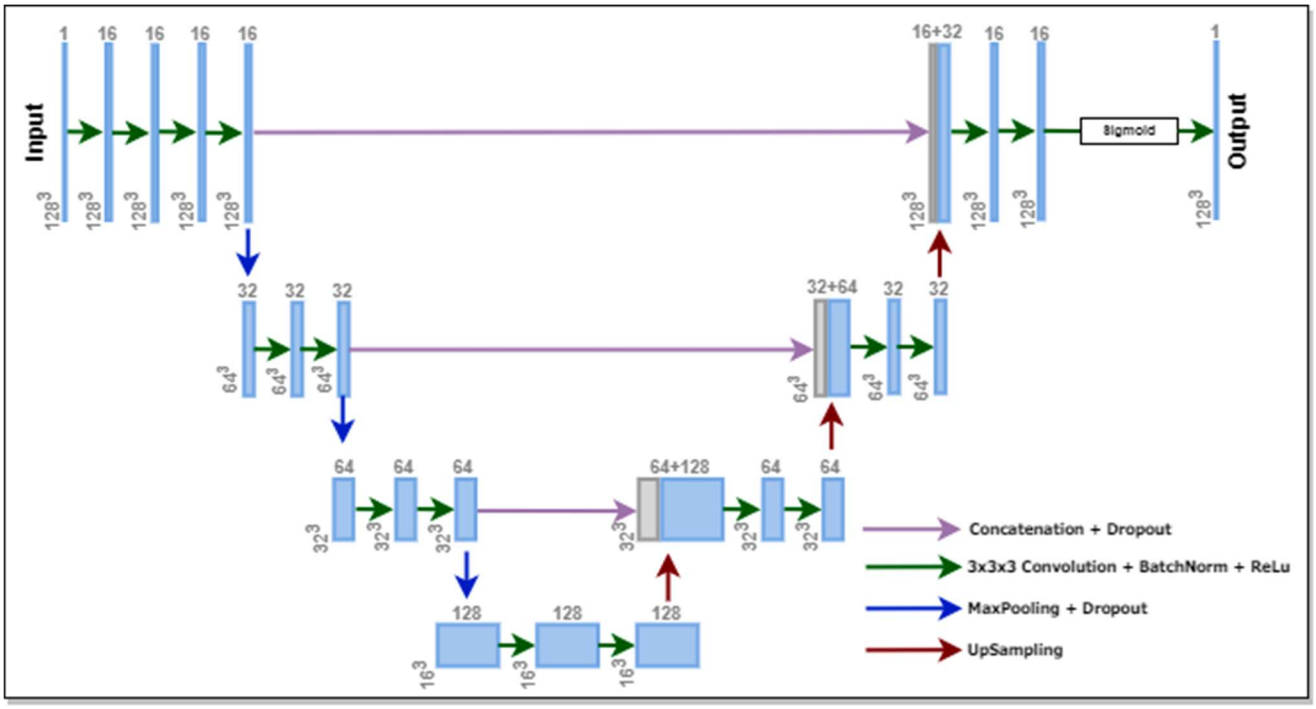


Figure 15: 3D U-Net Architecture [6]

Although the 3D V-Net model was able to train the model and get the prediction result, the model was poorly trained. Because of the loss value, dice loss, is around 1 throughout the training process. Since the highest loss value is 1, indicating the model is not learning. We still test the model with our input and got the images shown in Figure 16. The prediction mask is not ideal, because the bone parts are hollow. Luckily, the boundary of the mask is clear enough to find the angle we desired. We also overlapped the prediction mask in red onto the CT scan and found out that the prediction was quite accurate on the boundary.

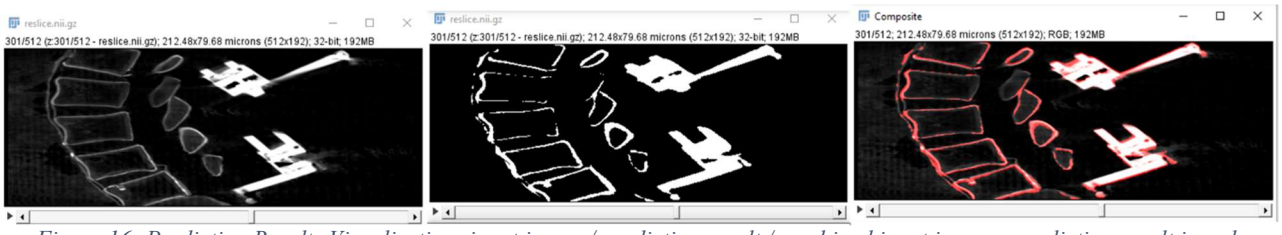


Figure 16: Prediction Results Visualization: input image / prediction result / combined input in gray, prediction result in red.

Therefore, we take the prediction mask as the input and perform the angle detection with steps shown below:

1. Change the coordinate to match the image to the CT scan.
2. For all the frames, find the frame with the largest predicted bone. This is our frame of interest.
3. Find the two largest connected components, which are the two trackers. Define the top one as tracker1 and the bottom one as tracker2 for reference.
4. Perform the same operations for two neighboring frames (center 2)
5. Get rid of the trackers for all five frames and average them, name the resulting image as bone image.
6. Detect the outline of the bone from the bone image using find Contours function.
7. Find the region whose y-axis is close to the center of mass of the tracker.
8. Find the two lines using hough_line_peak and calculate the angle. Figure 17 shows an example with two detected lines in thick white, which has an angle of 27 degrees.

Figure 17: Line detection from the predicted mask

The current methods will find the frame and the bone automatically based on the tracker position. While this method has lots of space for improvement. A better training model would result in more accurate predicted segmentation results. This could be achieved through varying training parameters, adopting a different training model, or increasing the training dataset. Moreover, a user interface could be added to allow users to pick any bone they want. Currently, the bone is selected based on the tracker position automatically. Third, this method could be adjusted to different types of trackers.

Functional BCA:

The functional BCA has only a few simple steps to create a model of the bone's edges. These are listed below in chronological order.

1. The BCA portion of the python script is run.
2. The stack of slices from CT scan is compressed into three 2D projections. This is done by summing all the slices along a particular axis. The three resulting 2D projections are displayed in Figure 18 below.

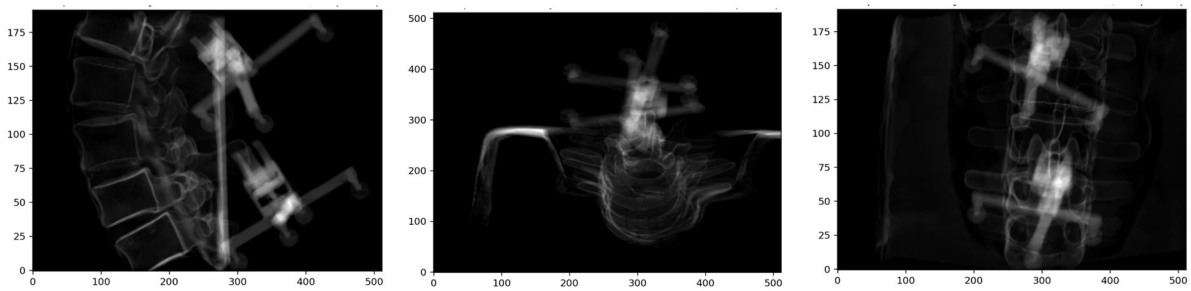


Figure 18: Three 2D Projections

3. Next, the leftmost of these projections displays in a window for the user to view. The pop-up gives three different planes for the surgeon to select from. So, depending on the corrective action required, the doctor will select the option most suited for his/her surgery.
4. After the plane has been selected, the user is prompted to create lines within the image by placing two points on the bone. Each time a point is selected, a text entry is required to confirm that you would like to place the point. Once two points are plotted, the line is saved for future use. These past two steps are displayed graphically below in Figure 19.

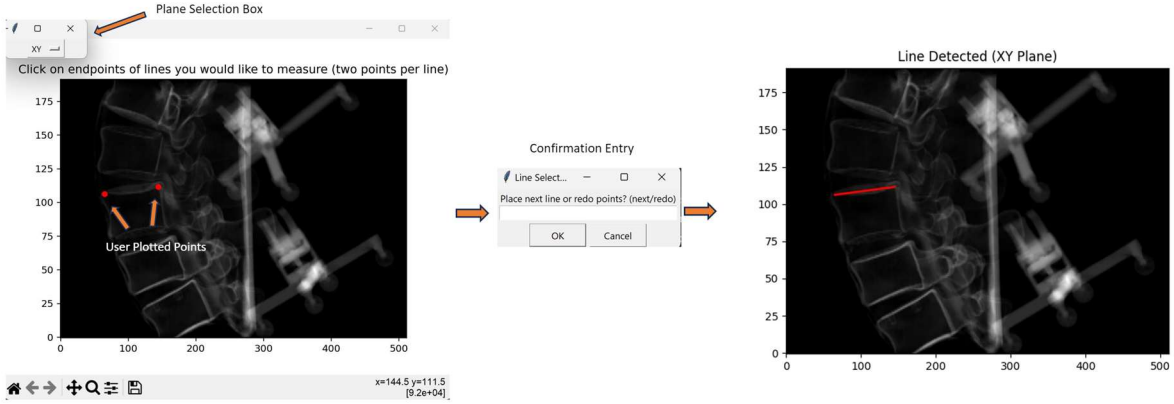


Figure 19: BCA User Input

5. The above step is repeated once more, as the surgery objective is to measure the angle between two bone edges. So, after this step is repeated, two lines will have been successfully saved. This terminates the BCA portion of the project.

Transformation Calculator:

Having got the tracker positions and orientations from the TCA and the bone position and angle from the BCA, now a relationship must be built between the two.

Pairing each Tracker with a line:

1. First, the coordinates of the trackers must be transformed to fit the coordinates of whatever projection was chosen.
2. The closes tracker to each line is chosen by calculating the midpoint of the tracker and using a simple shortest distance algorithm to assign each tracker to a line. The result can be seen in Figure 20.

```
Tracker Tracker_1 is paired with line 0 with coordinates: [[ 64.53225806 155.44805195]
 [145.0483871  147.65584416]]
Tracker Tracker_2 is paired with line 1 with coordinates: [[106.85483871  33.37012987]
 [183.24193548  46.87662338]]
```

Figure 20: Trackers assigned to closest lines.

Relating the Trackers to the Bones:

1. A transformation matrix is needed to place the coordinate frame of the trackers into the coordinate frame of the bone, so the output of the camera will be directly translated into the movement of the bones.
2. As can be seen in Figure 21**,** This can be done since the tracker orientation and the bone orientation as well as their positions is known.

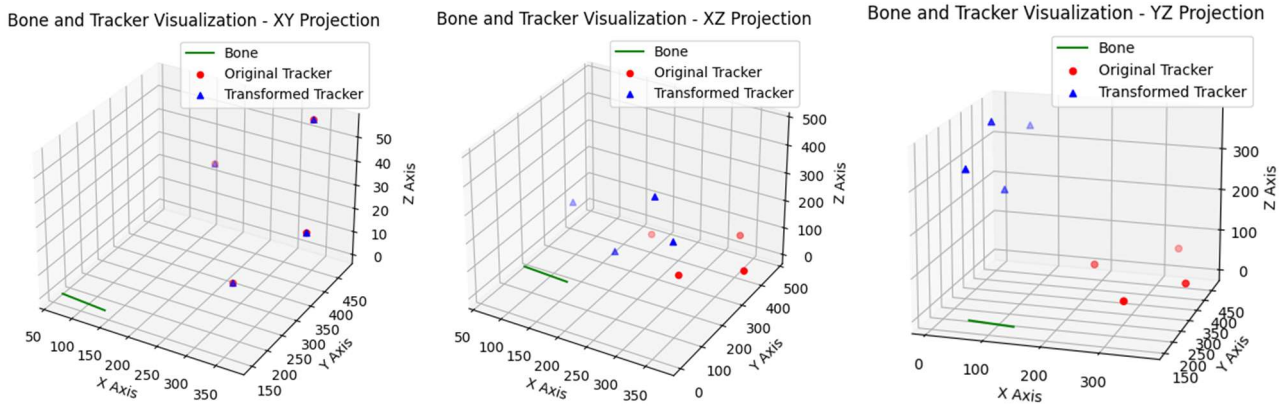


Figure 21:Shows sample transformation based on projection chosen.

3. After the transformation, the angle between the two lines can be found with simple geometry and can be displayed as can be seen below in Figure 22.

```
Angle between line 0 and line 1: 15.55 degrees
```

Figure 22: Printed angle between the two bones.

In the operating room, the camera needs to be directly interfaced with the script in order to get real time bone movement and angle measurements. This will require more testing to ensure the system is robust and efficient for a surgery setting.

Results:

As stated in the introduction, the objectives of the project were:

☑ 1. Create a program to classify the 3D trackers within a CT scan.
☑ 2. Create a program to classify bones within a CT scan.
☑

3. Create a program that uses the bone and tracker locations as input, and returns the linear transformation needed to relate the trackers to the bone.

Objective [1] was fully satisfied by the Tracker Detection Algorithm. The TCA uses a CT scan as

an input and returns eight $\mathbb{R}^3$ coordinates of the form: (x, y, z) . These coordinates are used as an input in a later objective.

Next, Objective [2] was satisfied by the Bone Detection Algorithm. Using the same CT scan as input, the BCA prompts the user to make selections based on the type of surgery performed. Combining the processed CT scan with the user inputs, the BCA returns two $\mathbb{R}^2$ coordinates (assuming the XY plane was selected) of the form: (x, y).

Last, Objective [3] was satisfied using the outputs from Objective [1] and Objective [2] as input into the Transformation Calculator. Using the eight coordinates from the TCA, two planes are created (one for each tracker). Next, the points created by the BCA are appended, such that two additional planes are created (one for each bone of interest). Using matrix algebra, a transformation is created such that each of these sets of planes can be transformed to align with one another.

Discussion

The objectives of the project were fully satisfied. The crux of the problem was to find a relationship between the trackers and the bones of the spine. To find this mapping our group requires only one CT scan to be taken prior to surgery. From there, a single program is run, which is a combination of the TCA, the Functional BCA, and the Transformation Calculator. This single program needs the CT scan, and a few simple user inputs, and the mapping between bones and trackers will be returned (as described in the 'Methods' section).

Going forward, the next natural step would be to integrate this transformation matrix with the NDI Camera software. Since the program produces a 1-1 linear relationship between the trackers and the bones, to integrate our program with the camera's software, the only required step is to apply the linear transformation to the real-time values the camera reads. Although a mathematically simple task, this step requires familiarity with the technical specifications and backend of the NDI software. To achieve this, the real-time values must be extracted from the software, such that the transformation could be applied. This finalizing step allows the user to determine all critical measurements within the NDI software.

Our team is very confident in the robustness of the current TCA. This portion of the program runs without error, and without outliers. However, as described in the section for the BCA, we see potential for improvement. Although the functionality of our current BCA is not compromised, some error is introduced when the user is prompted to select points to parameterize the bones. Since all doctors use a similar type of manual plotting to validate the spine procedure's successfulness, our team finds no issue using this feature to parameterize the bones. However, if the 3D V-Net Machine Learning Algorithm is adopted correctly, this could allow the user to simply select the name of the bone they wish to measure. This feature would immediately reduce the time it takes to run the program. With that, it would eliminate the human error introduced by plotting the points manually. For these reasons, our team agrees that the next iteration of this project should address this issue.

Conclusions:

Through the use of electromagnetic tracking system and developed software, the real-time tracking of bone positions and angles is within reach. The TCA proved to be the most successful part of the project as all the trackers were classified with high accuracy and robustness while maintaining efficiency. Furthermore, each individual balls of the trackers can be classified giving added flexibility for advanced coordinate transformation between the camera and the bones. This project successfully developed the skeleton of a real-time spine navigation system but there still remains work to be done in tying together all the parts and validating their robustness in the operating room. Once the system is fully operational, the length of spine alignment surgery will be significantly decreased as only one CT scan will be needed for calibration thus saving time in setting up and analyzing multiple CT scans. The future of such a system is very software intensive and hinges on the modern advances of machine learning that can further automate the system and add accuracy and robustness through model improvements resulting from the data of previous surgeries.

Deliverables:

The goals for the deliverables included a working spine navigation algorithm that enables orthopedic surgeons to assess a patient's misaligned bones during the corrective procedure. We were trying to meet all three objectives to reach the ultimate goal. We succeeded in creating a program to classify the 3D trackers within a CT scan, named BCA. The original plan for creating a program to classify bones within a CT scan was to use an interactive UI and ask the doctor to choose the line of interest in any direction. We shifted the plan to exploring machine learning algorithms to improve the accuracy and automatic the program to minimize user input. However, with the limitation in time, the machine learning algorithms were not ideal. Therefore, we switched back to functional BCA and planned to improve the machine learning BCA in the future. Third, we have created a program to match the trackers from camera reading to CT scan. The final report and website will still be a deliverable and complete project and leave many possible future directions and guidelines to improve and automate the design. All the code has been uploaded to GitHub for future reference.

GitHub Link: https://github.com/elkhashabs/Spine-Navigation-System-Capstone.git

Project Time-Line:

The project Timeline is displayed in Figure 23 and can be viewed more interactively through the following link.

Link: Capstone Project Outline.xlsx

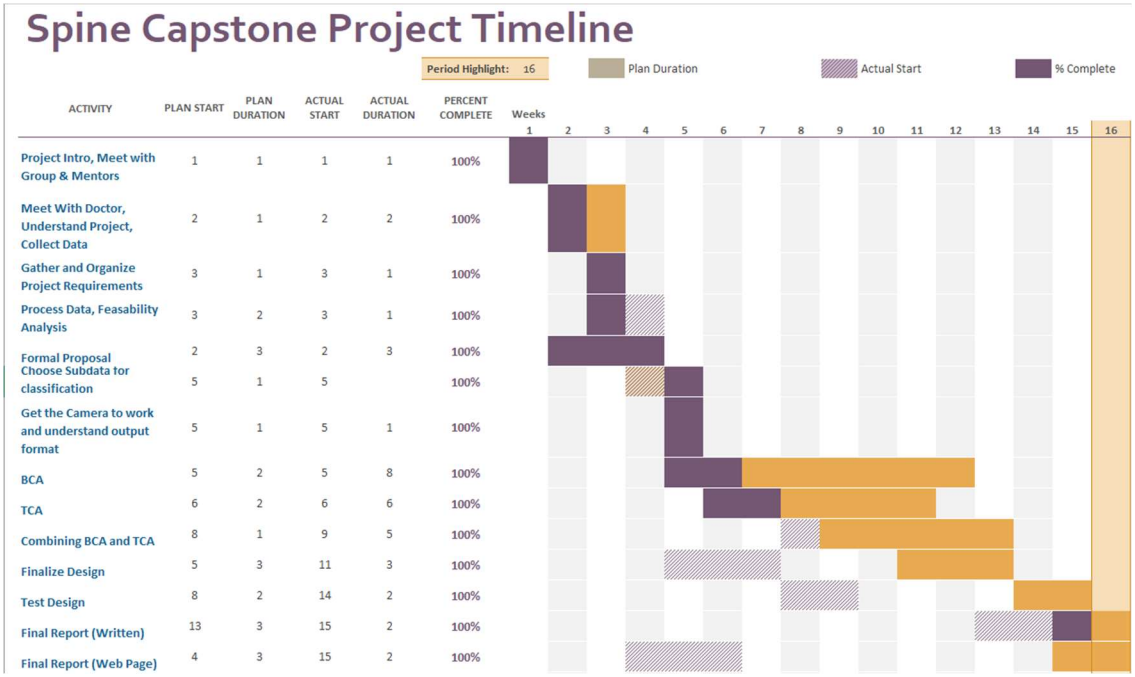


Figure 23: Shows the Timeline for the Project

References:

[1] Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation." *2016 fourth international conference on 3D vision (3DV)*. Ieee, 2016.

[2] Shim, Jae-Hyuk, et al. "Evaluation of U-Net models in automated cervical spine and cranial bone segmentation using X-ray images for traumatic atlanto-occipital dislocation diagnosis." *Scientific Reports* 12.1 (2022): 21438.

[3] Ronneberger, Olaf, Philipp Fischer, and Thomas Brox. "U-net: Convolutional networks for biomedical image segmentation." *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2015: 18th International Conference, Munich, Germany, October 5-9, 2015, Proceedings, Part III 18*. Springer International Publishing, 2015.

[4] Çiçek, Özgün, et al. "3D U-Net: learning dense volumetric segmentation from sparse annotation." *Medical Image Computing and Computer-Assisted Intervention–MICCAI 2016: 19th International Conference, Athens, Greece, October 17-21, 2016, Proceedings, Part II 19*. Springer International Publishing, 2016.

[5] Milletari, Fausto, Nassir Navab, and Seyed-Ahmad Ahmadi. "V-net: Fully convolutional neural networks for volumetric medical image segmentation." *2016 fourth international conference on 3D vision (3DV)*. Ieee, 2016.

[6] Luiserrador. “Luiserrador/ML_3D_Unet: Tensorflow Based Framework for 3D-Unet with Knowledge Distillation.” *GitHub*, github.com/luiserrador/ML_3D_Unet?tab=readme-ov-file#3d-unet-framework---includes-knowledge-distillation. Accessed 19 Dec. 2023.