Group 17: Automatic Assessment of Surgical Ergonomics
Final Report

Boyoung Zhao, Eric Han

Introduction

Surgeries in the operating room can result in surgeons adopting awkward postures, which can lead to poor circulation, muscle/joint pain, and increased fatigue. Studies have shown that surgeons spend over 50% of the time with their neck in flexion, which is often associated with tension myalgia. Suboptimal postures in general are associated with poor efficiency, and an optimized surgical setting can improve both efficiency and performance in the OR. The aims of this project are to

1. Automate a way for surgeons to see if they are spending time in dangerous positions, which saves time and resources so that surgeons can focus on surgery, and can quickly receive feedback about their posture.

2. Give ways for surgeons to improve their posture during surgery and show how much time is spent in dangerous zones - this allows surgeons to understand whether the awkward positions they are in are dangerous for themselves and understand risks to their own bodies.

Setup

For our setup, we first acquired an intel real sense RGBD camera (D415). Next, we used the Cubemos SDK to transform live videos of people into stick figure overlays. The whole setup process is attached at the bottom of our report. Figuring out the process was relatively difficult for us, and there are a decent amount of steps, so please refer to the bottom of the document for a full setup tutorial.

Technical Approach

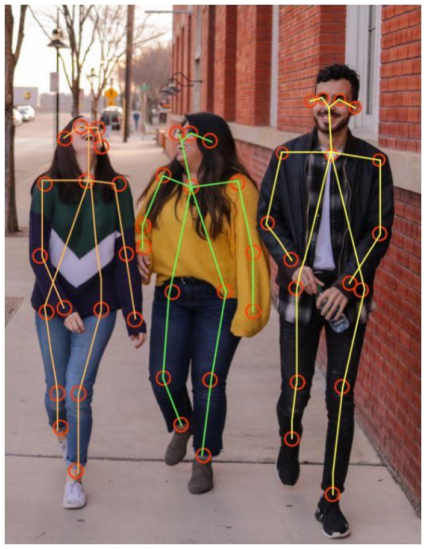


Figure 1: Depiction of Cubemos Skeleton Tracking

Our approach first required us to label each of the joints with its own unique joint ID. This allows us to keep track of each joint during each frame and handle cases where some joints may or may not be visible. In total, there were 17 of these joints, as shown in Figure 1.

Next, we wrote helper functions to assist us with the mathematical calculations. All of the functions were strictly unit-tested. These functions include:

float calculateAngle(
float x1, float y1, float z1,
float x2, float y2, float z2,
float x3, float y3, float z3);

This function uses the equation $\theta = \arccos\left(\frac{(\vec{BA})\cdot(\vec{BC})}{||\vec{BA}||||\vec{BC}||}\right)$ to calculate angles given 3 points, where vectors BA is found by subtracting the first and second points and BC is found by subtracting the second and third points.

std::vector<float> equation_plane(float x1, float y1, float z1,
float x2, float y2, float z2,
float x3, float y3, float z3);

This function computes the standard form equation for a plane $ax + by + cz + d = 0$ by calculating a,b,c and d using the equations below and returning the variables as a vector.

$$(1)\ \vec{AB} = (B_x - A_x, B_y - A_y, B_z - A_z)$$
$$\vec{AC} = (C_x - A_x, C_y - A_y, C_z - A_z)$$
$$(2)\ \vec{AB} \times \vec{AC} = (a, b, c)$$
$$a = (B_y - A_y)(C_z - A_z) - (C_y - A_y)(B_z - A_z)$$
$$b = (B_z - A_z)(C_x - A_x) - (C_z - A_z)(B_x - A_x)$$
$$c = (B_x - A_x)(C_y - A_y) - (C_x - A_x)(B_y - A_y)$$
$$d = -(aA_x + bA_y + cA_z)$$

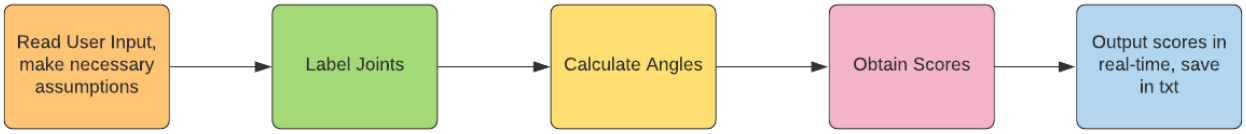
float plane_vector_angle(float x1, float y1, float z1,
float A, float B, float C, float D);

This function computes the angle between a plane and a point by the equation $\theta = \arcsin\left(\frac{|A \cdot x + B \cdot y + C \cdot z|}{\sqrt{A^2+B^2+C^2} \cdot \sqrt{x^2+y^2+z^2}}\right)$ where A,B, and C are taken from the standard form equation of a plane and x,y and z are the coordinates of the point.

float plane_plane_angle(float a1, float b1,
float c1, float a2,
float b2, float c2);

We calculated the angle between two planes using the equation $\theta = \arccos\left(\left|\frac{A_1A_2+B_1B_2+C_1C_2}{\sqrt{A_1^2+B_1^2+C_1^2}\sqrt{A_2^2+B_2^2+C_2^2}}\right|\right)$

We also wrote functions to read user input from a .txt file, which contains questions regarding the RULA and ROSA assessment. After the input is parsed, the main body of the code consists of a while loop which constantly detects and renders the skeleton while it is in view of the frame. In this while loop, each joint position is extracted, each of the angle measurements are calculated and their results are interpreted via the ROSA and RULA tables which we manually inputted. The score is then displayed to the screen and also saved in a text file. With regards to the joints which are not in view of the camera, we plan to output a small warning notification to the user which would inform them of which exact portions of the body are not in the line of sight of the camera, as scores without line of sight would be lower than expected.



Results

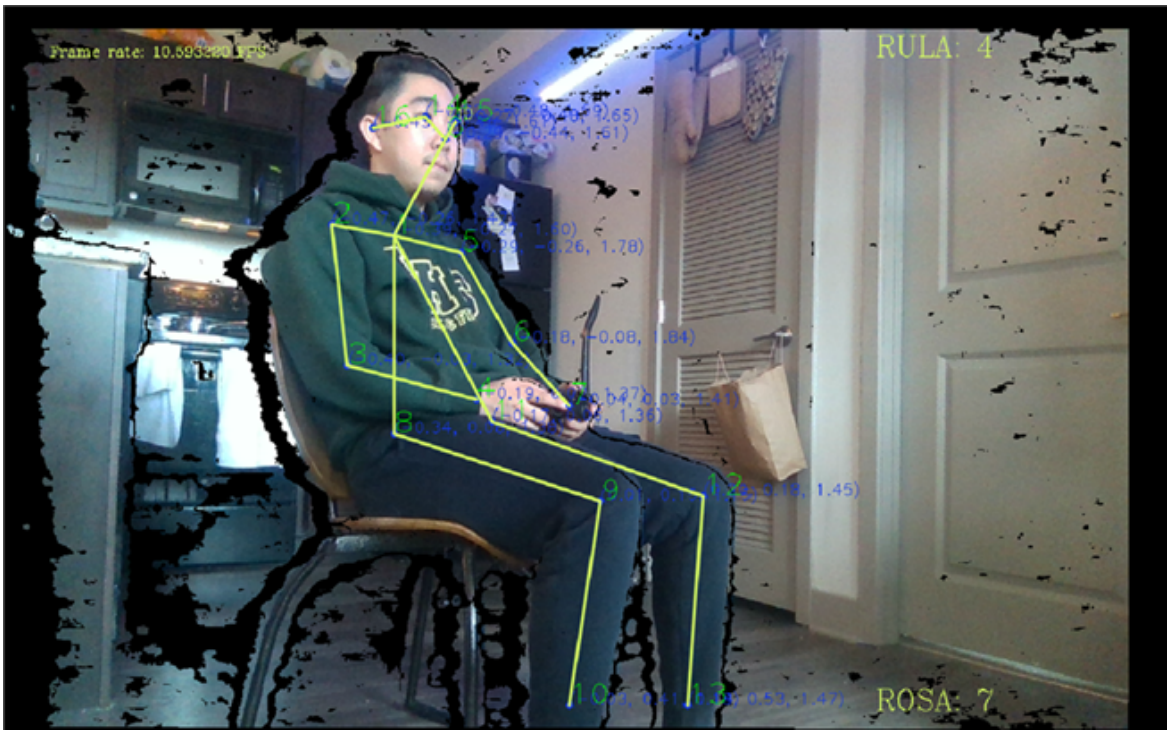


Figure 2: Screenshot displaying joint ID and position, real-time RULA (upper right) and ROSA scores (lower left)

To test if our results were accurate, we first tested each angle by assuming roughly 3-5 different poses which isolate the angle and comparing the real-time angle measurement outputted from the algorithm and the expected degree which we assumed the position to have. We did not have any stricter testing methods as it is extremely difficult to manually measure a depth angle and because the RULA and ROSA assessment angle thresholds are very forgiving. For example, the cutoffs for trunk angles are 0 degrees, 20 degrees, and 60 degrees. We found that our comparisons between assumed poses and outputted degree were accurate and moved on to assuming poses. We currently only have done 5 tests in which we compare the RULA and ROSA algorithm output of a frame and a manual RULA and ROSA assessment. For these 5

tests, we found that the discrepancy between expected and actual scores were always within 2 points. No statistical analysis has been done as there is not enough data points, but we do plan to continue testing in the future.

During testing, we discovered that when multiple people were in the frame, the score would fluctuate. This is because the outer loop of the algorithm loops based on how many distinct skeletons it detects, so the software would detect person A and output the score, then detect person B and output the score. We have yet to come up with a solution for this problem, but we will most likely output the skeleton ID next to the score so the user would know which set of scores correspond to his or her skeleton.

Discussion

Assumptions

To begin our discussion, there were a number of assumptions we had to make when running our program. This was mostly due to us not being able to calculate RULA and ROSA scores using only angles of joints. Our assumptions for both algorithms are listed below, as well as justification for why we thought these would be reasonable assumptions.

RULA

- **Step 1A: Assumed shoulders are not raised (+1 to upper arm score)**
    - Surgeons working on operations will tend to have raised shoulders when using tools
- **Step 2A: Assumed that one arm would be working across the midline or out to the side of the body (+1 to Lower Arm Score)**
    - Surgeons will most likely be reaching around for tools during operations
- **Step 3: Wrist scores: +1 would be if wrist was supported, +3 if not supported (we assumed the wrist would never be bent from the midline**
    - Change suggested by Dr. Galaiya and Dr. Formeister
- **Step 6: Posture was assumed to be mainly static (+1 to Muscle Use Score)**
    - Surgery is not a very dynamic action on the arms
- **Step 7: Assumed the Force Load score was <4.4 lbs (+0 to Force/Load Score)**
    - Surgeon tools are relatively light
- **Step 9: Neck flexion and extension treated the same (+4 for neck extension to +3)**
    - Surgeons will hardly ever have their head in extension when operating
- **Step 10: Assumed the trunk wouldn't be twisting (+0 to Trunk Score)**
    - Surgery does not require extreme trunk twisting- this is also difficult to recognize even with the SDK
- **Step 13: Assumed posture would be mainly static (+1 to Muscle Use Score)**
    - Surgery is not a dynamic action when sitting
- **Step 14: Assumed force load score was <4.4 lbs (+0 to Force/Load Score)**

    - No load should be placed on the neck/lower body

ROSA

- **Chair height and arm rests are adjustable**
    - Most modern chairs can be adjusted for height and have movable armrests
- **Arm rests are not on a hard/damaged surface and are not too widely spread.**
    - Arm rests are usually soft and can be adjusted so they are not widely spread

Assumptions can be modified as necessary depending on how the end user sees fit, but the underlying code would have to be changed as well.

User Input

In addition to assumptions, we also had a list of questions for the user that had to be answered before starting the program because they were too difficult for us to evaluate computationally. These questions are listed below, as well as a brief reason why we could not evaluate them.

RULA

- **(Wrist Score) Is the wrist supported? Yes: +1, No: +3**
    - Wrists are easily obstructed, so we modified the wrist score aspect based on feedback from Dr. Galaiya and Dr. Formeister
- **(Upper Arm Score) Is arm supported? Yes: 1, No: +0**
    - We could not find a way to figure out if an arm was being supported by something like a table or wall
- **(Wrist Twist) Will the wrist be twisted near the end of its range? Yes: +2, No: +1**
    - Again, the wrist is easily obstructed and some surgerys may require more twisting than others
- **(Leg Score) Are the legs and feet supported? Yes: +1, No: +2**
    - We could not figure out if the feet were touching the ground or a wall (similar to arm situation)

ROSA

- **Chair: Are the legs and feet supported? (answer can be taken from RULA) Yes: immediate 3**
    - Same as above, although answer is the same as the last question from RULA
- **Chair: Is there insufficient space under the desk/can you not cross your legs? Yes: +1**
    - We were unable to find a way to see whether there was insufficient space under a desk
- **Pan Depth: Is there around 3 inches of space between knees and the edge of the seat? Yes:1 No:2**
    - We could not measure out distance or recognize the edge of a seat

- **Back Support: Is there adequate Lumbar Support? Yes: check if chair between 95:110 degrees (1), No: 2**
    - Chair recognition was very difficult and required deep learning
- **Back Support: Is the backrest adjustable? No: +1**
    - Chair recognition was very difficult and required deep learning
- **Pan Depth: Is seat pan depth adjustable? No: +1**
    - Chair recognition was very difficult and required deep learning

Questions have to be answered in a text document beforehand, as mentioned in the technical approach. We attempted to make a GUI to make it easier for users to answer the questions, but the Cubemos software and GUI libraries did not work well together.

Future Work

The main course of action for future work on this project would be to reduce assumptions and user input as much as possible so that the program could be more automated. To do this, experience would be needed in Convolutional Neural Networks and Machine Learning. Although we got a better grasp of these topics, we did not know how to implement them.

Smaller changes that would be easier to achieve after reducing assumptions and user input would be to implement a GUI for user input, expand the program to apply to prerecorded RGBD videos, and to output a graphical and statistical analysis of how much time a surgeon is staying in certain score ranges. Further testing could also be done. One final possibility for future improvements would involve using two separate RGBD cameras from different angles to achieve a more accurate and complete skeleton.

Progress Evaluation

Overall, our plan for the project changed drastically from what we had in mind initially. Initially, we had planned on using 2 dimensional videos to calculate RULA and ROSA scores. We also had planned on using Matlab to run our program on pre-recorded videos. We ran into many issues, however, and ultimately ended up changing course midway through the project. We decided to use an RGBD camera and an SDK that allowed us to get a 3-D view of a person. Our program also only applied to live videos, and was done in C++.

A chart of our revamped deliverables is shown below:

Minimum	C++ code/documentation computing RULA and ROSA for a picture - 4/25
Expected	Process videos for RULA/ROSA scores, implement user input for wrist dependency/other possible dependencies, ask questions for more parts of difficult analysis - 5/4
Maximum	Implement a training algorithm with manually labeled images to calculate parts where user input is required - past 5/5

Figure 3: Deliverables

Because our project changed so drastically from what we initially planned, we had to work a long time to meet our expected deliverable. The maximum deliverable was not achieved, as it required the use of training algorithms and Convolutional Neural Networks, which neither of us were knowledgeable enough about to implement.

Our initial schedule was hardly adhered to, as many difficulties arose with the Matlab version of our assignment, and so we spent a lot more time reviewing literature than anything else to gain a better understanding of AI and CNNs.

In hindsight, if we had been able to switch gears earlier in the semester, we could have gotten farther into the project than we did, although the CNN and deep learning aspects may still have been too advanced as even after extensive research we struggled to implement them into the program.

Conclusion

Ultimately, although our project had more user inputs than we had originally expected and changed course greatly over the semester, our program lays a great foundation for future programmers with experience with CNNs and AI. We learned a lot about CNNs through papers, GUI creation, and how to plan for long term projects effectively. Furthermore, we learned about just how useful RGBD cameras can be and how advanced programs can detect whether a person is eating, sleeping, or performing some other sort of action.

Moving forward, we would like to make sure that any future users of our program can completely understand our code and the process of connecting the camera and Cubemos SDK,

which was relatively difficult for us to get working. Furthermore, the list of our assumptions and user inputs creates a starting point for anyone interested in advancing the program to make it more automated.

Lastly, we would like to thank Dr. Galaiya and Dr. Taylor for their guidance along the way, even through the rough times.

User Guide to using Intel-Realsense RGB-D camera for Skeleton Tracking

1. Download the zip folder from https://framos-my.sharepoint.com/:f:/p/c_scheubel/Em4eCGd-pctJnxkyG_pxRIgBcueMBoF5EuWtCYwGFEIdsA?e=2XZZZB by clicking the top left “Download” button

2. Go to https://www.intelrealsense.com/skeleton-tracking/ and purchase desired cubemos SDK license package

3. Extract download and run cubemos-SkeletonTracking application to install cubemos in C:\ directory.

4. Download CMake Installer from https://cmake.org/download/ (Win x64 Installer for Windows machine)

5. Run installer and check box that says “Add CMake to PATH for all users”.

6. Restart machine.

7. Download Visual Studio 2017 using the following link https://visualstudio.microsoft.com/vs/older-downloads/

8. Login with JHU credentials

9. Go to the downloads tab near the top and locate the Visual Studio 2017 link

10. Download Visual Studio Community 2017 v15.9

11. Select the Desktop Development with C++ workload and install

12. To activate the software for the first time run post_installation.bat script available in C:\Program Files\Cubemos\SkeletonTracking\scripts and enter the license key emailed to you in step 2.

13. If prompted to create a Visual Studio 2017 solution in the command window, press Y

14. Launch Visual Studio and sign in with JHU credentials

15. Go to File >> Open >> Project/Solution >> C:\Users\%Username%\Cubemos-Samples\Build\ALL_BUILD.vxcproj

16. On the right hand side, right click cpp-realsense underneath ALL_BUILD and select “Set as startup project”. Then navigate to cpp-realsense >> Source files >> cpp_realsense.cpp and open it.

17. Go to Build >> Build cpp-realsense

18. Go to Debug >> Start without Debugging. Camera should return joint locations and angles in real time.

19. Go to C:\Users\%Username%\Cubemos-Samples\Build\

20. Locate and replace files *cpp-realsense.sln, cpp-realsense.vcxproj, cpp-realsense.vcxproj.filters*, and *cpp-realsense.vcxproj.user* with the files inside team17.zip file.

21. Open the ROSA_questions.txt and RULA_questions.txt files and answer the questions with “True” or “False” (with quotations).

22. Repeat steps 15 - 18. Software should be fully operational.

23. Saved ROSA and RULA scores can be found in ROSAscores.txt and RULAscores.txt