Advanced Surgical Planning - Implant Recognition

Sarper Ertekin
ETH Zürich ITET
sertekin@ethz.ch

Orhun Görkem
ETH Zürich CS
ogoerkem@ethz.ch

Yuqing Huang
ETH Zürich CS
yuqhuang@ethz.ch

Laura Roduner
ETH Zürich RSC
rodunerl@ethz.ch

Abstract

This paper originates from a student project in co-operation with ETH Zurich and CustomSurg and aims to support surgeons during complex bone fracture surgeries using HoloLens to detect, track and label implants. Modern technology contributes massively to 3D Vision. Therefore, deploying Microsoft HoloLens in surgeries is not far-fetched, although relatively novel. There are many possible applications, while CustomSurg provided us with the opportunity to detect their custom implants and set the groundwork for HoloLens application in surgery, which we approached with off-device computation. You only look once version 5 (YOLOv5) deployed on a server communicates bidirectionally with HoloLens to send a captured image via TCP to the server, which processes the image and sends back information regarding the implant's bounding box and the label. Bounding boxes were obtained using HoloLens spatial mapping and several coordinate transformations. Our model is trained on synthetic data generated in Unity and yields almost perfect results on synthetic images while slightly less accurate for real images due to domain gap. This problem was addressed and tackled by manually adjusting the training data. Finally, Vuforia is used to compare our model to market solutions. It is also used to additionally track handheld implants which is not yet included in our data set but lacks robustness to sudden movements of the target object. The code for our work is available at: https://github.com/sarpermelikertekin/3dv-project-advanced-surgical-planning-implant-recognition.

1. Introduction

The *ETH* Spin-off *CustomSurg* [6] specializes in complex joint fracture surgery using modern-day technology. This project results from ETH's and *CustomSurg*'s joint interest and collaboration in terms of 3D Vision with HoloLens and is realized over a span of 3 months as part of the lecture *3D Vision*. *CustomSurg* is currently researching and deploying HoloLenses in complex bone feature surgeries in order to support surgeons making difficult decisions prior, as well as during the surgery. In the framework of this project we focus on choosing the correct implant, by recognizing it in HoloLens, tracing it, detecting the implant's type and rendering said type with the implant's bounding box in HoloLens. This way, surgeons can visually confirm that the present implant is truly the implant chosen for a particular surgery. The following sections of this report will outline the general methodology, technical details on the implementation as well as the results obtained.

For each fracture, there is an implant (see figure 1a) chosen based on its fit. These implants are typically standard size and shape, while sometimes adjusted by bending to fit a specific fracture. Each implant comes with an according guide, which enables the surgeon to pre-drill wholes in the patient's bone in order to screw the according implant into place (see figure 1b).

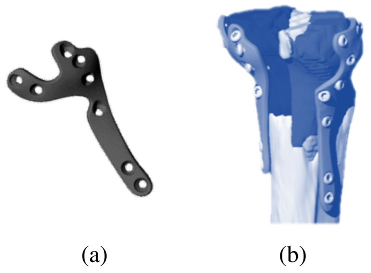

Figure 1: (a) Model of Implant. (b) Model of fractured bone with implant and screws.

3D Vision is a large field and has many possible applications. CustomSurg is known for using technology to reduce treatment time and improve surgical outcomes. We are given the opportunity to develop an application for HoloLens to assist surgeons before as well as during surgery and decided to contribute by implementing a general communication pipeline between HoloLens and a server, which among other things can be used to reliably detect implants.

We achieved this by carefully structuring sub-tasks, which will be elaborated in the following sections including corresponding results. For the work distribution, Orhun and Sarper are responsible for data generation; Orhun for Yolo training; Sarper for TCP communication; Yuqing for 2D-3D mapping of bounding box; and Laura for auxiliary research and report writing.

2. Related Work

In this section, we discuss related works on object detection in Microsoft HoloLens as well as on HoloLens 2 sensor data streaming.

2.1. Real-time Object Detection in Mixed Reality

Due to the headset hardware limitations, HoloLens can only run lightweight algorithms on-device. In order to use heavy object detection algorithms based on Deep Neural Networks (DNNs), many existing works chose to perform part of the overall computation in a cloud environment. Earlier work [4] presented a mixed reality system that can detect and track generic objects in a dynamic environment in real time. The system combines the HoloLens' processor with a cloud system equipped with high computational capabilities. The Cloud System is in charge to process *Region-based Fully Convolutional Networks (R-FCN)* algorithm to detect objects from a frame with the right compromise between speed and accuracy. On the other hand, HoloLens runs the Local System, which performs object tracking, feature extraction, and spatial mapping tasks.

Vuforia is a market solution that provides an augmented reality software development kit for creating augmented reality applications. It uses computer vision technology to recognize and track 3D objects in real time. Although *Vuforia* gives good performance for object recognition tasks, it is a "black-box" solution. Our work aims to provide a simpler and open-source pipeline that sends video data from HoloLens to the server, performs object recognition via *YOLOv5*, and sends back predictions to HoloLens for visualization. Moreover, we implemented object recognition in Unity using Vuforia Engine API for comparing the performances of our solution with the market solution.

2.2. HoloLens 2 Sensor Streaming

HoloLens 2 Unity Research Mode Streamer. [5] This repository offers Unity Plugin for accessing HoloLens 2 Research Mode sensors and video camera, and streaming them to desktop. We ran a Unity app using this Plugin and visualized the sensor data on desktop. However, we noticed that there are significant delays in the video camera streaming with this solution.

hl2ss. [2] This repository provides a HoloLens 2 server software and Python client library for streaming sensor data via TCP. The server is offered as a standalone application (appxbundle) or Unity plugin (DLL). Since we need to modify the server application to not only stream video data but also send prediction data back to the HoloLens, the server software from this repository cannot be readily used for our purposes. Therefore, we decided to implement our own TCP communication pipeline.

3. Method

3.1. Problem Statement

Since many implants look alike and it is not obvious which guide belongs to which implant, our goal is to make sure the surgeon uses the correct implant and the corresponding guide. In order to reach that goal we need to 1) recognize implants in a scene, 2) obtain a 3D bounding box and render it in HoloLens and 3) classify the implant's or guide's type, while also displaying the type in HoloLens. This way the surgeon can cross-check, whether the present implant is the one chosen for this surgery.

3.2. Communication Pipeline

We quickly decided on off-device computing, due to difficulties of deploying *YOLOv5* on HoloLens and eventual computational overhead leading to time delays when detecting the implants. Therefore we established a communication pipeline between HoloLens and our server.

On a high level, HoloLens captures a picture from its camera and sends it over TCP to a server to be processed and given as input to *YOLOv5*. When *YOLOv5* receives the image, it sends the class and position information back to the HoloLens as a string (multiple objects are supported), where this string will be further processed in order to place the bounding box and highlight the hologram of the detected implant.

3.3. Data set Generation

Since custom training is necessary to detect implants and guides, an according data set is needed. Manual annotation of real images could be time inefficient, therefore our approach is to generate a synthetic data set for training. We used 100 background images and placed the virtual models of our objects in front of them. The objects are randomly translated, rotated, scaled and colored (in gray scale). In total, we generated 3000 synthetic images for our training.

3.4. Training

The generated synthetic images are passed to the *YOLOv5* network. Their label files are prepared by converting Unity coordinates and sizes to *YOLO* format by normalizing. Training yields almost perfect results on synthetic data, whereas we encountered some domain gap between synthetic images and real images. Namely, the model learned to separate synthetic images from real images. Less training does not present better results. To tackle this issue, we annotated around 100 real images with physical models of implants and instruments and fine-tuned our trained model with them.

3.5. 3D Boxes in HoloLens

As soon as receiving the prediction results from the server, we perform a 2D-to-3D mapping. HoloLens Camera Stream makes the *ProjectionMatrix* (P) and *CameraToWorldMatrix* (C) available. Since the user's head is in the center of the HoloLens world coordinate system, we can write the projection matrix as:

$$P = \begin{bmatrix} f_x & 0 & p_x \\ 0 & f_y & p_y \\ 0 & 0 & 1 \end{bmatrix}$$

Therefore, it is straightforward to extract focal lengths f_x, f_y and center offsets p_x, p_y from the *ProjectionMatrix*. Using the pinhole camera model 2, we need to perform a sequence of coordinate transformations from the Image pixel coordinate system to the camera coordinate system to the world coordinate system. We compute the coordinate mapping as follows.

Given pixel coordinates (x, y) on the image, the ray direction in the camera coordinate system can be calculated by subtracting the center offset and divided by focal length, in each dimension.

$$\vec{r}_c = \begin{bmatrix} \frac{x-p_x}{f_x} & \frac{y-p_y}{f_y} & 1 \end{bmatrix}$$

The ray direction in HoloLens world coordinate system is calculated as $\vec{r}_w = C\vec{r}_c$, using the *CameraToWorldMatrix* (C). Finally, we need depth information to localize the exact position of the pixel in 3D. The method we are using is by shooting a ray in the direction of the center of the bounding box. The intersection point between the casted ray and the spatial mapping of HoloLens will determine the depth information.

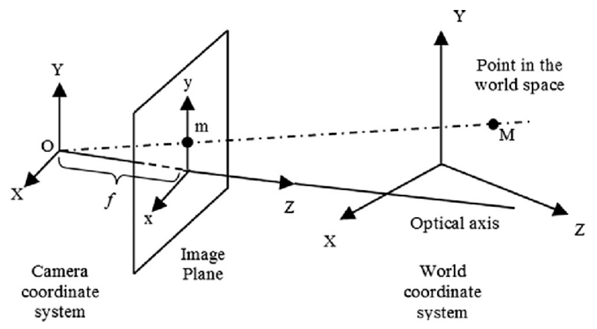

Figure 2: Pinhole Camera Illustration [1]

4. Implementation

4.1. Data set Generation

The synthetic data generation is implemented by ourselves in Unity. We prepared a scene with a platform containing the background image and placed the objects in front. For each image, we randomly picked active objects to be shown in the scene. The number of active objects differs between 0 and 5 in one scene, which was implemented purposely to improve training. Then, for each active object, a random color, rotation, lighting, translation, and scaling are applied. An example of our synthetic images is shown in figure 3.

4.2. TCP Connection

The communication is established bidirectionally, where the device containing *YOLOv5* acts as the server and HoloLens acts as the client. This part is entirely implemented by ourselves.

First, a picture is taken from the camera of the HoloLens, which is converted to a Byte Array to be sent via TCP. Since the image size is bigger than the chunk size we split the Byte Array into chunks and send every one of them separately, to fill the Buffer Array on the server side. As a chunk is received on the server side, we check if the Buffer Array is full. If not, a message is sent to the client, so that it keeps sending chunks of the same picture. As the Buffer is full, the image is reconstructed and given to the *YOLOv5* model to be processed. As output consists of a CSV file with every object present in the scene, with their names and positions, which will be sent to the client again. When this result is received, another picture will be taken to repeat this process.

4.3. Unity App - Bounding Boxes

Our implementation of visualizing bounding boxes in HoloLens world coordinate system is built upon the existing work HoloLensCameraStream for Unity [3]. The original repository provides a Unity plugin with useful functions to map image pixel coordinates from the HoloLens video camera to 3D coordinates. In our work, we present a Unity app

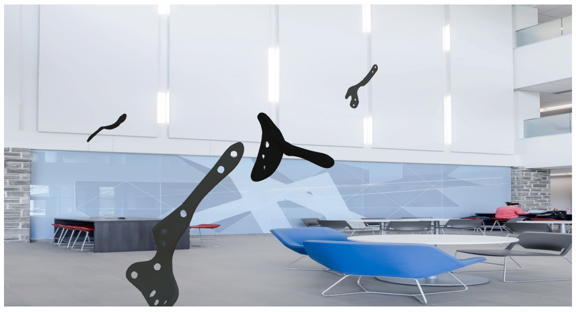

Figure 3: An example of generated synthetic image

that can 1) display in real-time what the HoloLens sees, i.e. HoloLens video stream, 2) convert bounding boxes specified in the image pixel coordinate system to HoloLens world coordinate system, 3) display the bounding boxes on top of the HoloLens video stream.

For our implementation, we first create variables that contain user-defined image pixel coordinates of a bounding box. Then these coordinates are converted to a ray direction in the HoloLens world coordinate system using *CameraToWorldMatrix* and *ProjectionMatrix*. Furthermore, a ray is shot from the HoloLens origin to the center of the bounding box, and we attempt to find where the ray intersects with the spatial mapping of HoloLens. Finally, we use the distance of the intersection point as depth to place the bounding box in 3D. Unfortunately, we had some issues with the spatial mapping of HoloLens, therefore our current application shows the bounding box on top of the image plane without the depth information.

4.4. Unity App - Implant Highlighting

Given as an incentive by the representatives of CustomSurg, we decided to also implement the functionality of highlighting the hologram of the detected implant(s), complementary to the bounding boxes. This will help surgeons to visualize quickly which type of implant they have picked.

To achieve this, we created another script, which handles the CSV file received from the server and extracts a list of the names of the implants. We map these names with the actual Game Objects and change their material to the highlighted material (see figure 4). This implementation including the user interface of the App is done by ourselves.

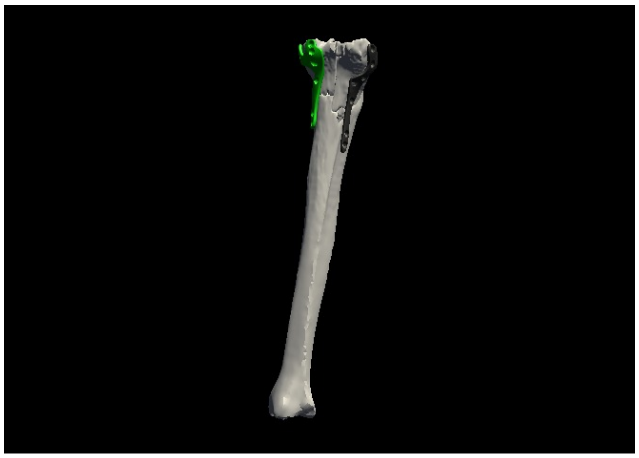
Figure 4: Highlighted implant on a virtual bone fracture model

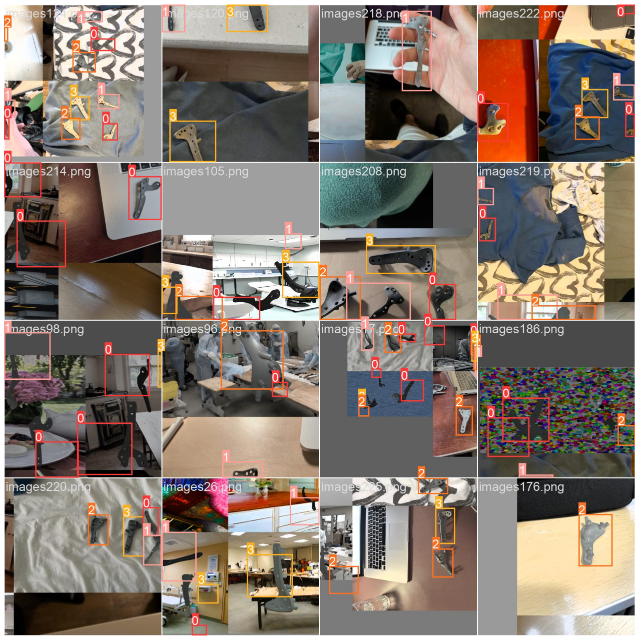
Figure 5: Some examples of detection

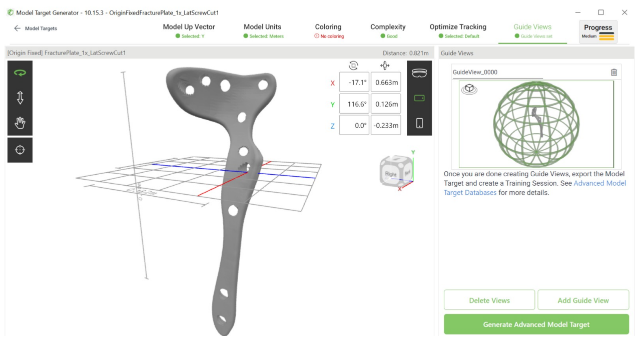

Figure 6: Implant shown in the Model Target Generator

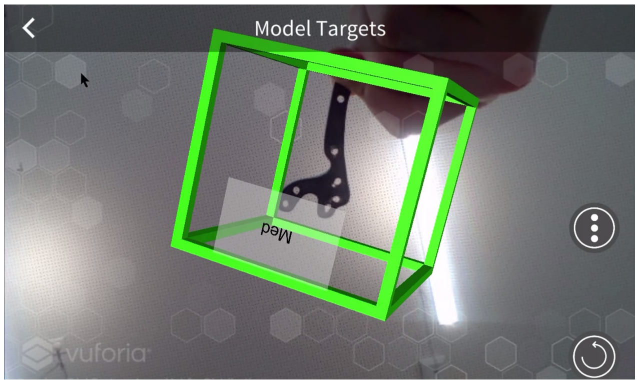

Figure 7: Target Model of the implant in Vuforia

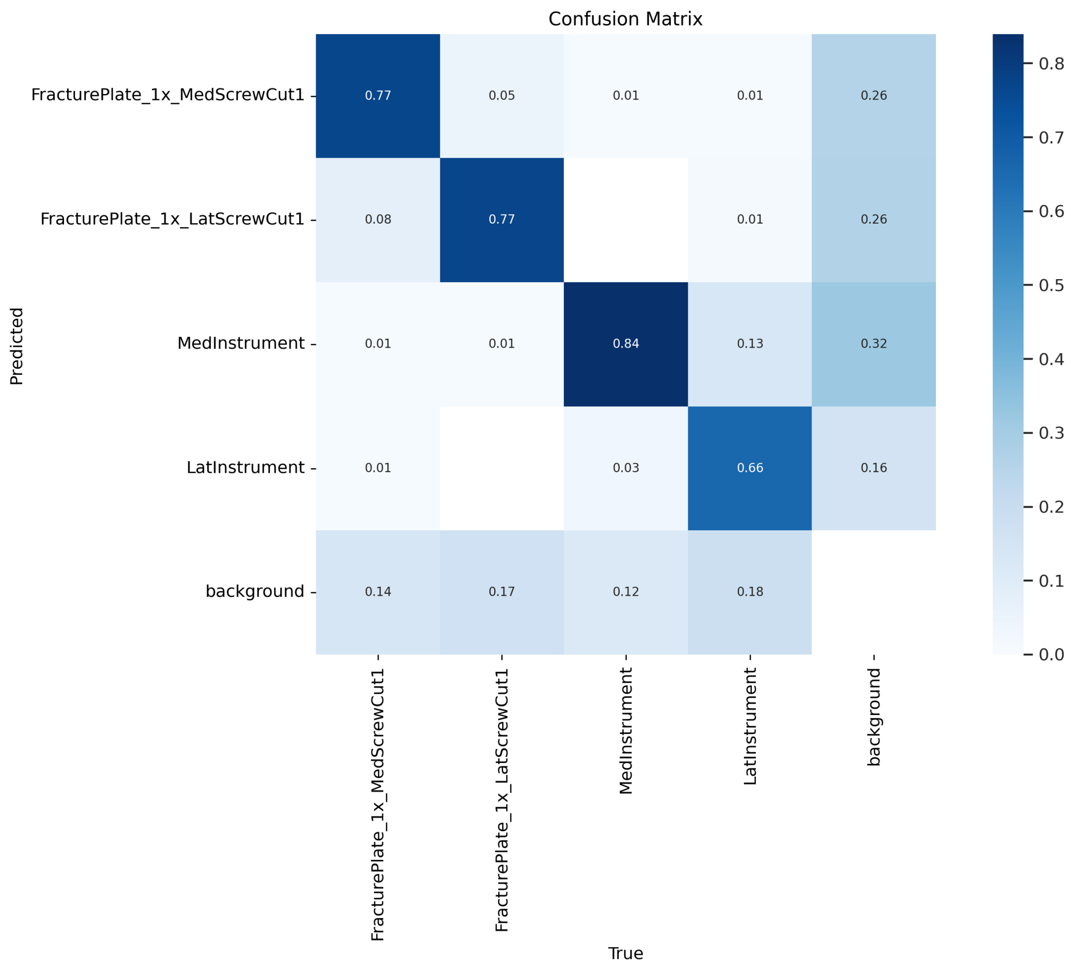

Figure 8: Confusion matrix of training

5. Results

Regarding the training data and performance of training, the *YOLO* training revealed almost perfect results on synthetic data, whereas real images suffered from domain gap. Then, we under-fitted the synthetic data and fined tuned the obtained model with manually annotated real images. We saw that in our results precision is higher than recall, which means that when an object is labeled, it is generally correctly labeled, and a random background is not labeled as an object. However, the model sometimes misses to label our objects. Quantitative results as well as some randomly selected training samples can be seen in figure 5 and 8.

Detection delay is an important factor as the end goal is to achieve real-time detection on HoloLens. In our solution, the inference duration is around 0.13s. The performance of tracking in total including network latency is around 2 FPS. Note that by tracking we mean the number of detections as we do not use any other method for tracking.

5.1. Vuforia

Finally, to compare our results, we implemented a Vuforia scene for both handheld devices and for HoloLens. In this setting, we used *Model Target* feature from Vuforia, which enables the user to track 3D models given a CAD model in their *Model Target Generator* application.

To use this feature, one must upload the CAD model of the desired object into the *Model Target Generator* application (see figure 6 and 7). Afterwards, the model can be configured to be recognized from different angles. After creating the *Guide Views*, which enables the model to be recognized, one can create a data set from different models. Upon creating a data set, we import it into Unity and the 3D Models will be recognized by Vuforia.

As it is a professional app, created for business application, the tracking and mapping works well, even though the model is not very robust to the sudden movements of the tracked target.

6. Discussion and Conclusion

As outlined earlier, we rely on *YOLOv5* for training and Vuforia for comparing our model's performance. This decision was based on their suitability for our task. We also considered two existing repositories to work with HoloLens, one of which was discarded due to extensive delays in the video streaming, while the other repository was modified to our needs.

As aforementioned, synthetic data yields almost perfect training results in contrary to real data, due to domain gap. Fine-tuning the model with manually annotated real images performs well, but is cumbersome and inefficient in implementation and not generalizable to large data sets and more complex applications. We also saw that precision is larger than recall, while both are to our satisfaction. Keeping in mind, that we support surgeons detecting whether the intended implant is present or not, precision is more important for this setting. A mislabeled implant or guide could be mistaken as intended for the surgery and falsely used on a patient, with severe consequences. While failing to detect the implant at all would still be undesirable, but with less severe consequences, since this error is easily detectable.

Off-device computing proved to be a valid implementation for this task since it is much faster than Unity deployed directly on HoloLens, which produces significant delays of around 20 seconds. Moreover, there is no need for actual tracing.

Vuforia is a standard and professional application and therefore performs well. Furthermore, it allows us to detect handheld devices, which is not implemented in our solution. In contrast, our model is more robust to sudden movements.

Future work would first of all include displaying proper 3D bounding boxes. We laid the groundwork for that, but unfortunately could not entirely finish within the framework of this project. As mentioned before, our model was not trained to detect partially covered implants, which would be a simple but effective addition, since in real-life scenarios implants are often handheld. This can be achieved by generating a new data set, that includes partial implants. A similar practical addition is learning whether dull gray implants can be detected on metallic gray background since operating rooms are often equipped with sterile metallic surfaces or casings. Future work should also include increasing time efficiency and potentially implementing a separate model for tracking. Both continuations would be extensive and simply don't fit this work's body.

In conclusion, this work lays the groundwork for CustomSurg to use HoloLens as support for surgeons in complex bone and joint fracture surgery. It performs well in detecting, labeling implants and corresponding guides. Our work also presents a reliable and generalizable framework for object detection, labeling and displaying in HoloLens using off-device computing.

References

[1] Y. Deldjoo and R. E. Atani. A low-cost infrared-optical head tracking solution for virtual 3d audio environment using the nintendo wii-remote. *Entertainment Computing*, 12, 2016. 3

[2] J. Dibene and E. Dunn. Hololens 2 sensor streaming. *arXiv preprint arXiv:2211.02648*, 2020. 2

[3] EnoxSoftware. HoloLensCameraStream for Unity. 3

[4] A. Farasin, F. eciarolo, M. Grangetto, E. Gianaria, and P. Garza. Real-time object detection and tracking in mixed reality using microsoft hololens. *VISAPP*, pages 165–172, 2020. 2

[5] C. Gsaxner. HoloLens2-Unity-ResearchModeStreamer. 2

[6] Wyssraumdesign. CostumSurg Website. 1