FindIt

Sefi Albo, Bar Albo

Supervisors: Yaron Honen, Boaz Sternfeld

Idea

FindIt is an HoloLens application for finding an object in home environment. The idea is to help us find missing stuff at home by tracking the objects around us and remember their locations.

Introduction

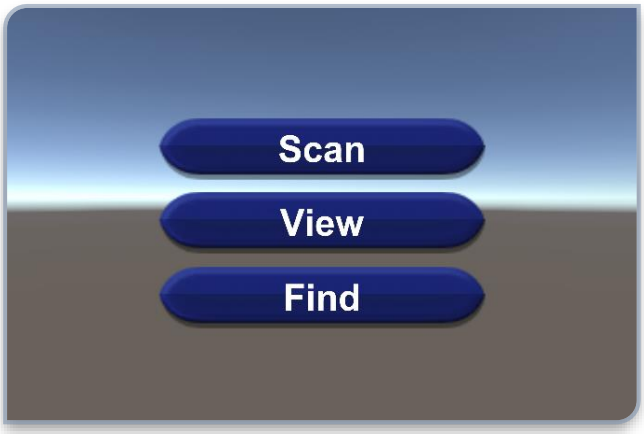
We developed an HoloLens application using Unity 2017 – an engine for developing games and application for variety of different platforms.

HoloLens is an augmented reality headset developed by Microsoft, that lets you see, hear, and interact with holograms within an environment such as a living room.

Application User Guide

The app has 3 modes:

- Scan – scan the room to find objects.
- View – view the scan results and adjust.
- Find – find the objects and track changes.



Scan

This mode's purpose is to initialize the app knowledge about the objects the users wants to track. The app is taking photos which will later be processed by Object Detection Model to detect the objects in the room. There are two ways to take photos:

- The default one is saying: Take Photo
- By Saying "Change", we can use the second mode which is Gazing. It means that every time the user gaze upon a location for about a second, the App will take a photo.

View

This mode's purpose is to view the scanning results and adjust them. In this mode, the user can see all the detected objects and their location. When the users Tap on a box, it will tell the user the object name.

If the object detection missed an object, the user can look at it and say "Add <object name>". A new box will appear at spot the user looked at.

Find

This mode is main usage of our App. When users say "Find <object name>", the box will appear at the exact same location, as the object. Also, a navigation system will guide the user to its location.

Another important feature of Find is tracking. To change an object location or remove it entirely, When the user is picking up an object the HoloLens will track its position. If an object was out of view for a long time, a message will say "<object name> was removed". If the tracker detected a new position, the message will be "<object name> location changed".

Implementation

Object Detection

In the “Scan” phase, the user can take photos by saying “Take Photo” or by gazing for a second at the same location. When the user says “Stop”, the HoloLens waits for the server to process the images.

Each time the HoloLens takes a photo, it also provides two different matrices: Projection matrix and World matrix that are used to convert any 2D position in the image to a 3D position and vice versa.

Translating between 3D position to 2D position can be done using only just these matrices (3D Math).

Translation between 2D position to 3D position is harder because we can’t know the depth of the exact point, only the direction. So, to know the exact location, we use Spatial Mapping.

Object Detection – Spatial Mapping

The HoloLens can understand the world around him using the Spatial Mapping component. This component creates an invincible mesh that covers the entire room surfaces such as: floor, tables, walls etc.

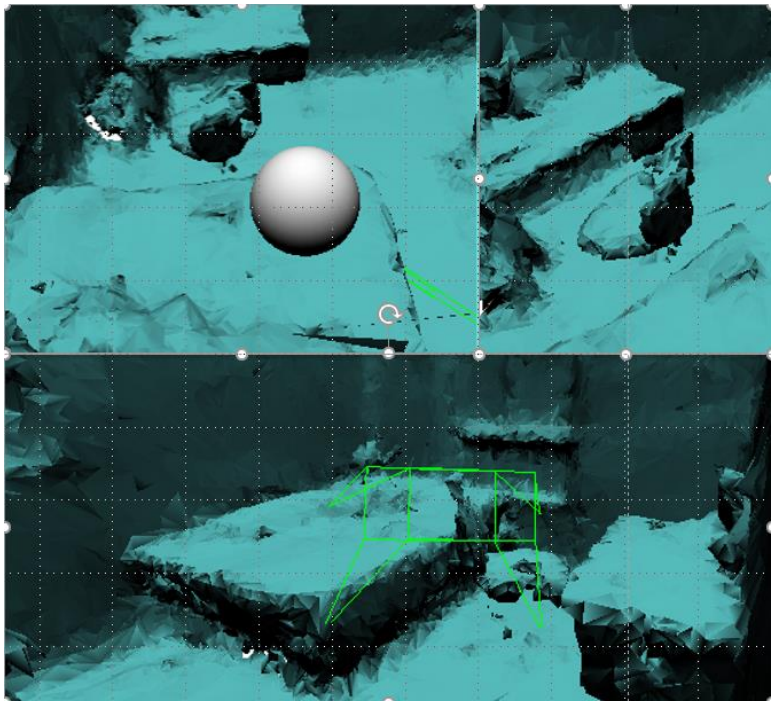
When the user gaze at a certain point, we can use the usual Unity Ray Cast to hit the spatial mapping layer and get the real-world 3D position the user gazed at (hit point).

Using 3D Math, we can translate 2D position in an image, to a 3D direction in space (not real position, because we can’t know the depth). We then find the exact 3D location by using Unity Ray Cast with Camera position as origin, and the direction we found.

Spatial Mapping – Example:

Here we How the HoloLens sees the room in real Time.

Taken from the Device Portal.



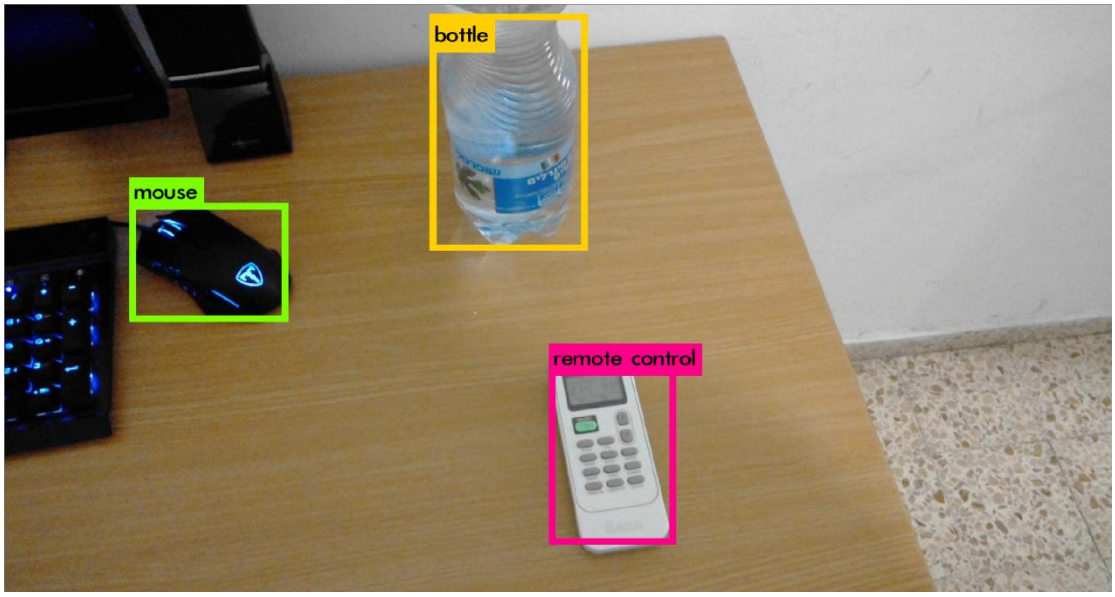
Object Detection – Detector

We used YOLO V2 – Deep Learning architecture for object detection. YOLO is running on a remote server, which is written in Python.

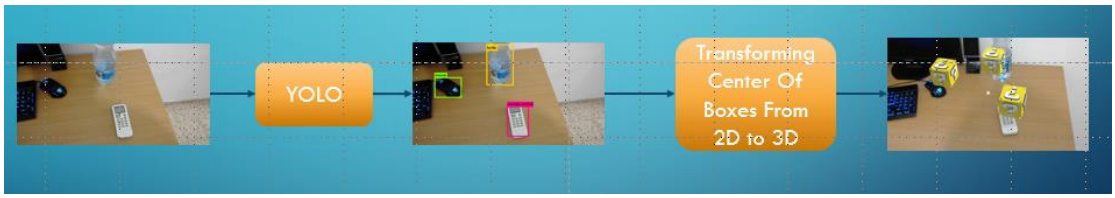
The server receives series of images and then run YOLO on each one. Then it sends back for each image, the list of boxes it found. For each box we have the object's name, 2D position in the image and confidence.

When the HoloLens receives the boxes (from the detector), it converts the 2D positions in the images to 3D positions (as we explained) and places markers at these real-world positions.

YOLO V2 example



Object Detection – High Level Flow



Here we can see the flow:

When the user says "Take Photo" a new image is passed to the YOLO, then we get bounding boxes for each object. After that, we transform the center of boxes from 2D to 3D and the output is the marker boxes on each of the objects.

Object Detection – Persistent Positions

Because every time we restart an app, the coordinate system changes, there is a need to set a fixed location. We used World Anchor Store to do that. Using world anchors, the objects will stay the same place even upon restarting the HoloLens. We also use the File System to save the list of objects we found.

Object Detection – View Mode

In the “View” mode, the user can see all the objects the app found and click on each one. Clicking on a box will says its name (for example: cellphone).

Because the object detection can sometime fail, we added an option to add objects manually.

The user needs to gaze at the desired position and say “Add <object name>”, then a box will appear at the gazed location.

Object Detection – Find Mode and Navigation

In the “Find” mode, the user can say “Find phone” and then an arrow will appear that points to the phone’s location.

Using the camera position and the object’s position, we can calculate the direction and projects it on the screen panel.

Object Tracking

Object Tracking is one of the key features of this App.

We use Object Tracking to track objects in real time to adapt to changes in the room.

For example, the user has picked an object and move it to somewhere else or removed it entirely from the room.

Tracking an object consists of various stages. The first stage, “Trigger” is done by the HoloLens.

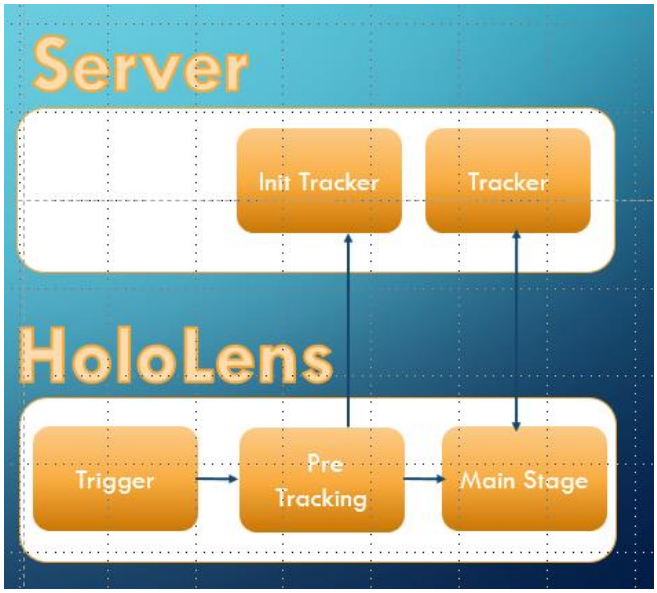
First, we need to determine which object do we track. So using the spatial mapping, we can see if the user gazes at an object. If the user gazes at an object for couple of frames, and is closed to it, we assume he is going to pick it. Then we start tracking that object and “Trigger” stage is over.

Although this condition isn’t sufficient to determine if user is indeed picking up an object, it can save a lot of unnecessary computation. For example, the user can’t change the phone location if he is far away from it.

After the Trigger found an object to track. We take the first picture and converts the object 3D position in space into a 2D position.

We then send the 2D position and the first image to the server

Object Tracking – High Level Flow



- Trigger - looks for a new object to track by tracking the user's position and nearby objects.
- Pre Tracking - takes the first picture, converts the object 3D position in space into a 2D position in the first picture and sends it to the server.
- Init Tracker - initialize the tracker using the 2D position and the first image.
- Main Stage - takes pictures and sends them to the server.
- Tracker - The server updates the tracker with the images it received and sends back the object's new 2D position according to the tracker.
- The HoloLens converts them into 3D coordinates and updates its state machine accordingly.

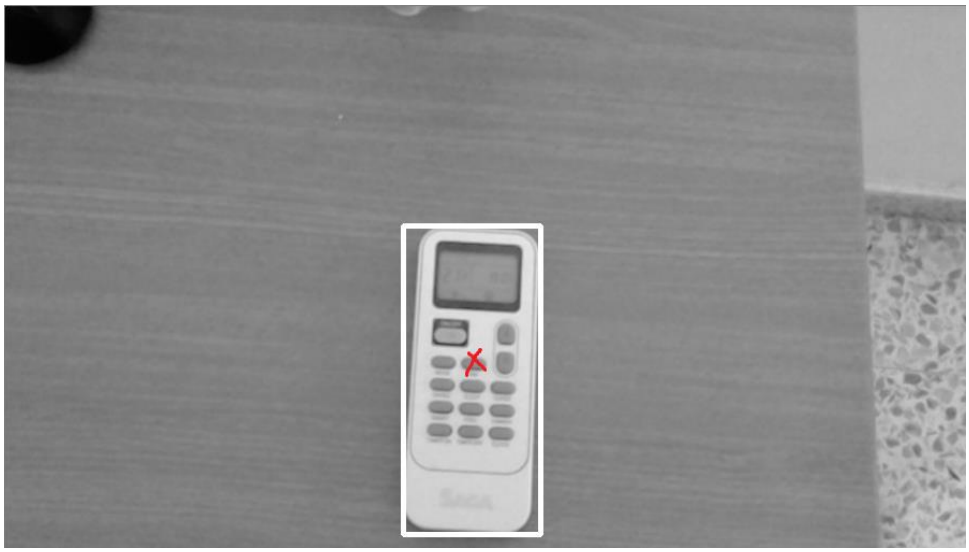
Object Tracking – Init Tracker

The tracker algorithm needs to be initialized with a bounding box of the object we want to track.

Therefore, when the server receives the 2D position, it tries to find a bounding box that is close to this position to initialize the tracker.

To find the closet bounding box we first find the contours in the image using OpenCV.

Example:



Object Tracking – Tracker

Two of many difficulties of implementing the tracker:

HoloLens can only take pictures in a very low frame rate (about 5 FPS), so images suffers from motion blur and “instant” movements of objects.

Our tracker also needs to be a good reporter. Some trackers track the object very well, but they can’t say when the object is leaving the view.

We tried many different trackers and we found CMT to do well with both these challenges.

CMT was implemented using OpenCV on the server.

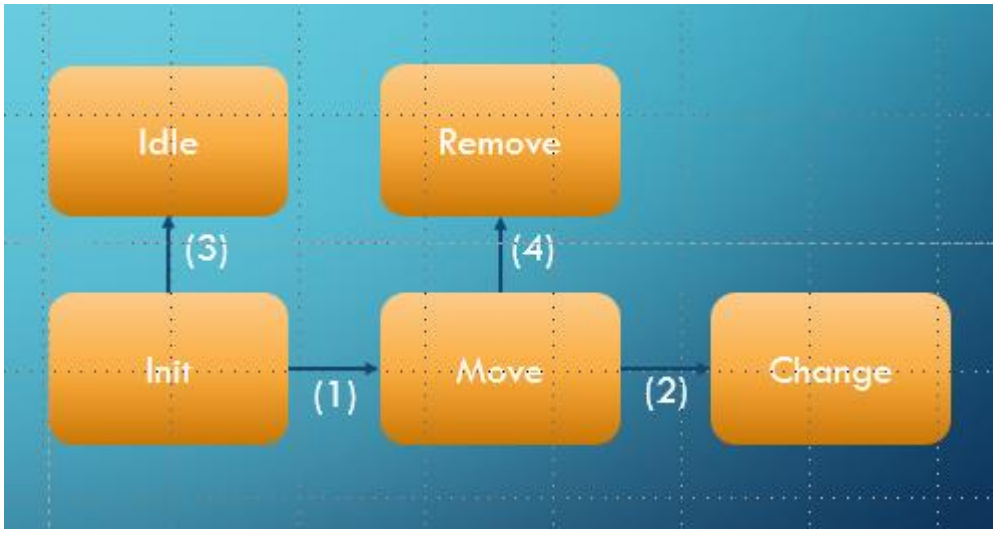
Object Tracking – Main Stage

The tracking – In the following pictures, we send the picture and save the matrices for later. In order to increase FPS, each time we take a picture, it's pushed to a queue.

Then an external thread process this pictures, sends them to server and wait for a response. That's how the communication doesn't interfere with the App.

The server receives an image, update the tracker and sends back the center of the bounding box the tracker found. If no bounding box is found, it can mean two things: Object is out of view, or the tracker just fail to detect it in the image. Either way, the returned point is (0,0).

Object Tracking – Main Stage – State Machine



(1) – the object position must change from its initial position by some threshold.

(2) - the user must be closed to the object and the last positions haven't changed (meaning it's not moving anymore) so a new position is set for the object.

(3) – the object was out of view for some frames, but it wasn't moving so tracking is done and the object is "Idle".

(4) – the object was moving but suddenly became out of view for enough frames, so we assume the object was removed.

In all the states: Idle, Remove, Change the tracking is done, and we go back for searching for a new object to track.

How to Install:

1. Install Python 3.5
2. Install OpenCV 3
3. Follow instruction to get YOLO: https://github.com/AlexeyAB/darknet
4. Download the project: https://drive.google.com/file/d/1PjCLk5Gyj676IpHLmCKh5eE4QTcOf6xl/view?usp=sharing
5. Replace the yolo_changed_files with the real files
6. Rebuild YOLO (darknet_no_gpu.exe)
7. In server folder, run server.py and logger.py (change IP address to your IP)
8. Open findit.unitypackage on Unity and Deploy on HoloLens.