Object Detection with ZED Mini and Unity

Aurora Bertino
Robotics Engineering Master Course
University of Genova
Genova, Italy
Email: s4399133@studenti.unige.it

Sara Romano
Robotics Engineering Master Course
University of Genova
Genova, Italy
Email: s4802844@studenti.unige.it

Chiara Saporetti
Robotics Engineering Master Course
University of Genova
Genova, Italy
Email: s4798994@studenti.unige.it

***Abstract*—**
Object detection is a core problem faced in many robotics scenarios such as grasping, manipulation, localization and AR/VR applications. The rise of machine learning approaches, together with the steady increase of the available computational power, and the low cost and abundance of camera sensors, made Computer-Vision techniques one of the most researched field in robotics and automation. However, the industry is still adapting to this new trend and many devices (i.e. cameras) and frameworks still lack any form of object detection. The aim of this project is to obtain a vision-based object detector integrated in Unity and able to generically use neural networks in the ONNX format on the video output of a ZED Mini stereo-camera. To achieve this, we used the Barracuda package and adapted it to work with the ZED framework inside the Unity ecosystem. Finally, we tested our system with the Tiny-Yolo v2 network. Project details can be found on GitHub[1].

I. Introduction

In recent times, object detection and pose estimation have gained significant attention in the context of robotic vision applications. [1]. As a longstanding and fundamental problem in computer vision, object detection aims at automatically determining the existence and spatial location of certain instances of objects in the camera images [2]. It can effectively help autonomous and intelligent systems such as machines or robots to perceive and understand the world better [3] by automatically analyzing digital signals from cameras.

Cameras and robots are technologically ”ready” to do object detection, but algorithms to do so are not always present in the SDKs. The idea of this project comes exactly from this problem, since Stereolabs provides object detection only for the ZED 2 Camera, while all other devices of the ZED family do not have this capability. For this reason, we proposed a system that performs real-time object detection with the ZED Mini Camera and we integrated the camera on Unity.

The ZED Mini is a stereo camera that provides high definition images and accurate measurement of the environment depth. It reproduces the way human vision works, using its two “eyes” and, through triangulation, it provides a three-dimensional understanding of the scene that it observes. In order to use it, StereoLabs offers APIs (to retrieve the camera image, set the parameters, get the depth information etc), and a ZED Unity plugin that allows the integration of the ZED with Unity.

Namely, our contribution in this research is to define a solution to correctly take the camera frames in the correct format, import the neural network TinyYolo v2 in Unity using Barracuda (a neural network inference library) and finally to retrieve the 3D coordinates of the object expressed with respect to the camera frame. In the end, the resulting bounding boxes and the object 3D coordinates are shown on the Unity GUI.

In Section II we list we give a brief description of the instruments that we used, and some of the possible approaches to Object Detection on Unity. In Section III we describe our work, and in Section IV our results. Finally, in Section V we discuss our conclusions and considerations for further development.

July 5, 2021

II. Materials and Methods

In this section, we briefly discuss the hardware and software used in this project, the possible approaches, and our chosen technique.

Hardware and Software

As hardware we used the **ZED mini** from StereoLabs [4], which is a stereo camera which has a maximum resolution of 2560x720 pixels and a FoV of 90○(HxV). The baseline of the camera is 63mm, which is similar to the human average eye distance. The ZED SDK [5] provides a depth map obtained through stereo triangulation. Its depth range is from 0.1 mt to 20 mt [6]. The laptops that we used to test the results have the following specifications. The *first computer* has an AMD Ryzen 7 4000 series processor and a Geforce GTX 1650 graphics card. On this laptop we were able to test the SVO files (30 fps) on Unity without any problems. However the software crahsed when we tried to use the real ZED Camera (60 fps). The *second computer* has a Intel Core i7 10th generation and a GeForce MX 330 graphics card. Using this laptop we were not able to test our program neither with a SVO file (30 fps) nor with the real camera.

As software we used several platforms and frameworks. The ZED SDK is integrated in Unity (which was our graphical engine of choice) by the ZED Plugin for Unity [7].

[1]The repository can be accessed at:
https://github.com/sararom15/ObjectDetection-ZEDMini-TinyYolov2-Unity

Unity is a cross-platform game engine used to create three-dimensional and two-dimensional games, as well as interactive simulations and other experiences [8]. It also allows the use of pre-trained Neural Networks through the Barracuda package.

Barracuda package is a lightweight cross-platform neural network inference library for Unity [9]. It can run neural networks on both the GPU and CPU, and it is possible to choose the platform by specifying it from the code. The Barracuda neural network import pipeline is built on the *ONNX (Open Neural Network Exchange) format* and supports some neural architecture models , among which we opted for a pre trained Neural Network Tiny Yolo v2.

Tiny Yolo v2 is a real-time neural network for object detection trained on the Pascal VOC dataset. It runs at 40 fps and detects 20 different classes. Although it is a simplified and lighter version of Yolo v2, TinyYolo v2 is a good trade-off between accuracy, running time and memory storage: this is why it is widely used in computer vision applications and embedded systems [10].

The network takes as input a square image with 3 color channels (416x416x3 array) and returns the object detection results as a 13x13x125 array. In particular, this array is composed of 13x13x(5(1+4+20)) elements: YOLO divides up the image into a grid of 13 by 13 cells; each of these cells is responsible for predicting 5 bounding boxes. A bounding box describes the rectangle that encloses an object and each one has four values *(x, y, h, w)* and a confidence score. For each bounding box, the cell also predicts the class probabilites. This works just like a classifier: it gives a probability distribution over all of the possible classes (Tiny Yolo v2 has 20 classes). [11], [12].

Methodology

In order to perform Object Detection on Unity with the ZED Mini we evaluated some options:

The first possible path was to use the ZED 2 APIs for Object detection on Unity. As we previously stated, Real Time Object Detection algorithm on Unity already exists , but it requires a ZED 2 camera [13], while the original ZED and ZED mini do not support this feature. This interaction is not available due to the technical properties of the different cameras: the ZED 2 gets higher performances than the ZED Mini [14]. A possible approach can be to modify the existing framework to work with the ZED Mini as well. However, this option is likely unfeasible, for example because their 3D app requires an high depth range, and the ZED 2 has a way better range than the ZED-Mini (due to the bigger baseline).

Another option was to use *ZED's integration for Tensorflow*. Using the ZED-Mini, Real Time Object Detection can be performed with Tensorflow1 with visualization of the boxes around real-world objects detected in OpenCV [15]. The approach can then be to look for a possible integration of the ZED-Mini with OpenCV and test it. However, when analyzing this option, we found that there exist few OpenCV integrations with Unity, and they are either a paid version or a very limited free version (which is not very used nor documented).

Therefore we also evaluated the option of using *Unity with Vuforia*. With this package, objects can be tracked in real time based on their 3D scan made via the ZED-Mini. This option, although very interesting, was really different from the others, since it is more AR oriented. Moreover, Vuforia is transitioning into a closed license system.

Finally, the last option we considered was using *Unity with Barracuda / ML Agents*: Real Time Object detection can be performed with Barracuda with visualization of the classified objects in Unity. The proposed work consists in checking if it is possible to use the ZED mini as input device and adapt its output to make it compatible with the neural network. Even if the Neural Networks supported by Barracuda are very limited, this option seemed the most appropriate and feasible to follow.

III. Implementation

Our implementation is split in four phases. First, we fetch the ZED camera stream in Unity (Sec. III-A). Then, we import a neural network using Barracuda (Sec. III-B) to detect the objects in the scene. Afterwards, the depth is estimated from stereo triangulation (Sec. III-C), and finally visualized on the video stream in Unity (Sec. III-D). In the following sections, each part will be discussed in detail.

A. Using the ZED in Unity - fetching the video stream

After importing the camera into Unity, our goal was to retrieve the Zed webcam texture, which is the texture onto which the live video input is rendered and on which the object detection result is written. There are a few ways to do so, and in the following we are described the ones we tried.

Starting from the typical approach when using cameras on Unity, we looked for the camera devices available with *WebCamTexture.devices*. By printing them we saw that the ZED camera appeared to be the second device as we expected, but the texture resulted empty. This is probably because the ZED camera needs to be used only through its available API.

Then, we tried following a suggestion that we were given online on the Github forum; using *TextureEye()* from *ZedRenderingPlane.cs*. The function returns the texture of the real world generated from the ZED. Once created, the ZED SDK automatically updates it with each frame/image the ZED capture. However, also in this case the texture resulted in being empty.

Finally, we tried to retrieve the webcam texture by using methods from the *ZEDManager*, which is the main code of the Zed Unity plugin and contains all the functions to work with the Zed camera. It provides two methods for texture retrieval. One of them is *CreateTextureImageType()* which provides the texture on the GPU only . To get images with the CPU, the *RetrieveImage()* function must used. It retrieves an image texture from the ZED SDK and returns an output of type *ZEDMat*, where the latter is a memory pointer used to store image data.

Code structure

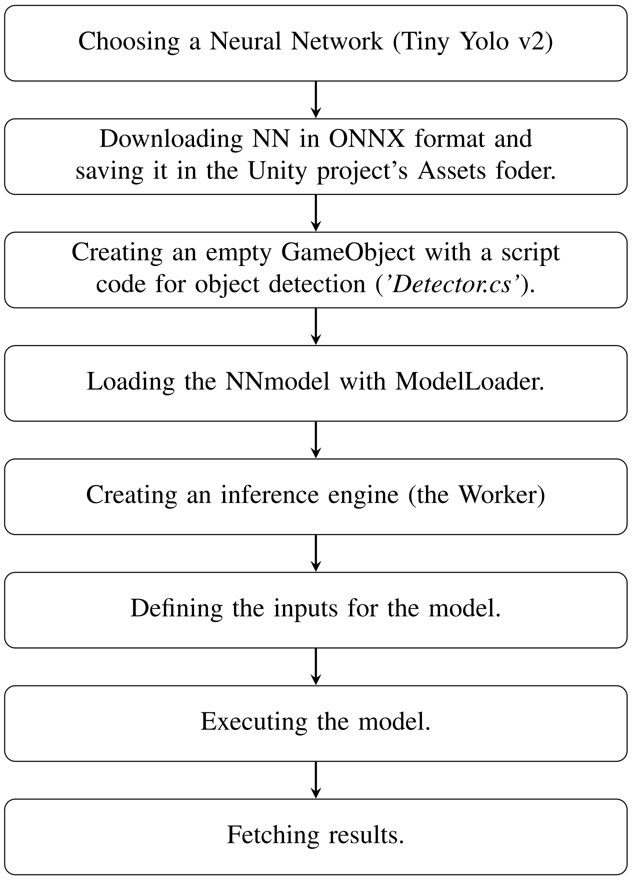
This part of the code is inside the *CameraData.cs* file that we described before. Inside the *start()* Unity function, we retrieve the ZedManager and the ZedCamera from the ZED namespace.

Inside the *Update()* function we define the *ZedMat* to store the image by specifying that we want to store the mat data on the CPU. Successively, we retrieve the image and marshal copy the ZEDMat into a byte array. This is finally loaded on a Texture2D by using the method LoadRawTextureData() with format BGRA.

B. Importing a Neural Network in Unity using Barracuda

Barracuda workflow

As already stated, Barracuda allows to execute a pre-trained neural network in Unity [16]. The steps that we used to integrate Barracuda in the project are summarized in the following Flow Diagram:



Choosing a Neural Network (Tiny Yolo v2)
Downloading NN in ONNX format and saving it in the Unity project's Assets foder.
Creating an empty GameObject with a script code for object detection (*'Detector.cs'*).
Loading the NNmodel with ModelLoader.
Creating an inference engine (the Worker)
Defining the inputs for the model.
Executing the model.
Fetching results.

More specifically, each step performs the following operations:

- Deciding which neural network to use (choosing from the neural networks that Barracuda supports). We opted for Tiny Yolo v2.
- Downloading the neural network in ONNX format and saving the file in the Unity project's Assets foder. It will be imported and show up as an asset of type NNModel.
- Creating an empty GameObject and assigning a C# script to it (in our case it is called *'Detector.cs'* and it will be explained later in detail). This file will contain the code for the object detection.
- Adding a public NNModel field to the CSharp script and assigning reference to the asset via editor UI. In the code, the model is loaded with ModelLoader.
- Creating an inference engine (the Worker) responsible for breaking down the model into executable tasks and scheduling them on GPU or CPU.
- Defining inputs for the model. They can be provided both as sole Tensor object (assuming the Model has only one input) or as a dictionary of name and Tensor pairs. We needed to use the tensor with the following structure: (batch, height, width, channels, source data, name).
- Executing the model. It can be done synchronously (Execute()) on the GPU, asynchronously (ExecuteAsync()), or in a blocking way (ExecuteAndWaitForCompletion()). The YOLO model does seem to work better with ExecuteAsync() than other methods. [17], [18].
- Fetching results (with PeekOutput()).

Code structure

In order to execute the aforementioned steps in the specific case of Tiny Yolo v2, we created the script *'Detector.cs'* which takes as input:

- the NNModel "Tiny-Yolo v2", imported as an asset in the project folder.
- the txt file containing the label names, imported in the project folder as well.

The code reads the label names from the imported txt file, loads the NNModels and creates a Barracuda worker. Then, it takes the input Texture2D from the camera and transforms it into a Tensor object. After that, the model is executed. As stated above, the Neural Network divides the input image in 13x13 grid cells: for each of them, it computes 5 bounding boxes with their respective *(x,y,h,w)* values (whose meaning has been defined in the previous section), the probabilities for all the classes and the box confidence score. The minimum confidence score that we accept is 10%.

It then compares the bounding boxes found in a similar area and keeps only the ones that have highest confidence. The metric that it uses is the Intersection Over Union, which means that it decides which boxes to keep by looking at the ratio of intersecting areas / union area. The threshold that we chose for IOU is 0.3, which means that if two boxes intersect for up to 30%, the one with lowest confidence is discarded. The code then maps the boxes coordinates to the right pixels (since x,y are defined with reference to the single cells and not on the whole image) and sorts the bounding boxes into a list according to their confidence score. The array of the final bounding boxes (each one containing dimension, confidence and label) is then set as output. Because of the IOU, this list may also be composed of one element only.

C. Depth Sensing

To achieve a 3D object detection, it is important to retrieve to the Z-dimension of the detected object and Stereolabs offer

APIs for that [19]. There exist two possibilities to access the depth value.

The first one is via the *Depth Sensing API* [20]. In this case the distance value (Z) for each pixel (X, Y) in the image is computed by triangulation and stored into a matrix (depth map). The distance is calculated from the back of the left eye of the camera to the scene object. The procedure to retrieve information is very similar to the one computed to retrieve the image for the Texture2D. In addition, since we wanted to only retrieve the depth of the detected object, we just saved the distance of the pixel corresponding to the center of the bounding box upon detection.

The second one is via the *Point Cloud API*. This API is used to get the 3D point cloud with (X,Y,Z) coordinates and RGBA color [21]. The point cloud stores its data on 4 channels using 32-bit float for each channel. The last float is used to store color information, where R, G, B, and alpha channels (4 x 8-bit) are concatenated into a single 32-bit float. When measuring distances, the 3D point cloud could be used instead of the depth map. The *Euclidean distance* formula allows to calculate the distance of an object relative to the left eye of the camera. The procedure to save Point cloud information is very similar to the one for depth map.

D. Result Visualization in Unity

Image retrieval

When a new project is initialized in Unity, the new sample scene is by default created with a Main camera GameObject that corresponds to the computer camera. The usage and visualization of the ZED Camera on Unity is instead allowed by the ZED Unity plugin through the Mono and Stereo camera rigs that can be found in the prefabs folder. These prefabs are custom AR cameras that replace the regular Unity Camera in a scene. The ZED Rig Stereo prefab contains both the left and right video sources for passthrough AR. When adding the ZED Rig to the hierarchy, it will appear at (0,0,0) with -Z representing the forward direction facing the camera. Instead, the ZED Rig Mono has the Camera-Left only, which contains a Frame holding the video source of the left camera. After importing the ZED (stereo or mono) camera, the Main Camera should be deleted from the Hierarchy to avoid interfering with the embedded camera of the prefab [22].

In our project, we performed object detection using only the Left Camera, meaning that we only imported the *Mono Camera prefab*. The pose estimation output is therefore computed with respect to the left camera frame.

Results visualization on Unity GUI

In order to plot and write data on the Unity screen, a Canvas GameObject must be added in the Hierarchy. Canvas is the area in which all User Interface objects must be placed. Since we are working with Images, we have to display a Texture2D for the UI System, therefore we have to put a RawImage GameObject inside the canvas. To combine the canvas with the ZED visualization, we need to specify: Inside the Canvas object: *Render Camera = Camera_Left*; Inside the RawImage object: *Texture = ZEDView Native Size*.

Code structure

After preparing our Unity scene with the camera, we attached a C# script to the Camera object (*CameraData.cs*). The most important steps of the resulting script are: accessing the camera to retrieve its output in a code-compliant format (Texture2D), applying the correct scaling and cropping (also rotation if needed) to its output and finally sending the modified Texture2D to the script for object detection. How to retrieve the image from the ZED is specified in the following section. Then, upon receiving the results from it, the last step to perform is to draw the rectangles around the detected objects and to show on screen (via Canvas) some useful data such as label, accuracy, position in space.

IV. Results

We initially tested our project on recorded files in SVO format. This was due to the fact that it was not possible to access the ZED Mini in the laboratory often, due to the Covid-19 situation. However, in the end we were also able to test the work on the real ZED Mini camera. In our code we opted to print the box with the highest confidence only, and print the other boxes (if existing) only if they have a confidence higher than 40%. Along with the box we also print the label with the corresponding confidence level, and the 3D coordinates of the center of the object with reference to the left camera frame. Note: we printed the text in different locations for easier readability.

In the two examples below (Figs. [1, 2]) we performed object detection on a bottle and a chair. In these examples the bottle is detected with a high confidence while the chair has a lower confidence. The difference may be due to the fact that the chair is not the only object of the scene and there is no white background.

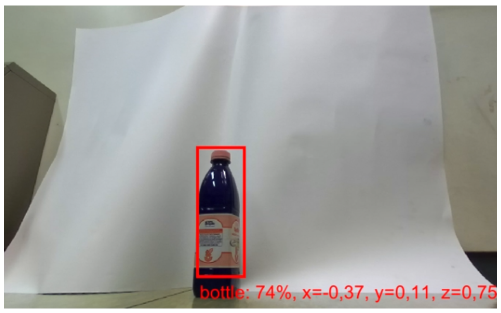


Fig. 1. The Object detector in action. X and Y coordinates are referred to the horizontal plane of the left camera of the ZED mini, while Z is the depth (in meters) obtained through the triangulation of the two views. The percentage at 74 is the likelihood of a correct classification inside a dictionary of 20 classes.

During the testing phase of our project, we recognized that the program is slow when it works in Real-Time: it needs a few seconds to detect and frame the object in the scene. Therefore, if the frames change quickly, there is a risk of not

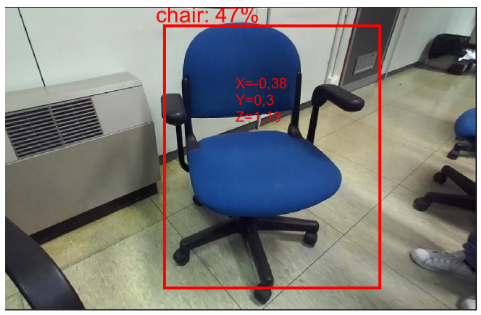


Fig. 2. The Object detector in action. X and Y coordinates are referred to the horizontal plane of the left camera of the ZED mini, while Z is the depth (in meters) obtained through the triangulation of the two views. The percentage at 47 is the likelihood of a correct classification inside a dictionary of 20 classes.

getting a perfectly defined bounding box of the object. This may be due to the fact that the camera works in Unity at 60 fps, while Tiny Yolo v2 Neural Network runs at 40 fps on the CPU. However, results improved a lot by using videos in SVO format recorded with the ZED Mini at 30 fps.

These results are still a good accomplishment if the objects that must be detected move at low speeds or are steady and positioned on a surface. For example, if our work had to be used on a manipulator that needs to pick different objects in front of it, the velocity should not be a problem.

V. Conclusion

The goal of this project was to perform object detection on Unity by using the ZED Mini camera from StereoLabs. By considering its architecture, the ZED Mini mimics how our own two eyes perceive the world around us: this allows to turn VR headsets into high-end AR pass-through devices. Precisely for this reason, performing object detection directly on Unity could be a first step to perform Augmented Reality with ZED Mini and AR headsets. StereoLabs has made available an API for Real Time Object detection on Unity, however it does not work with the ZED-mini, but only with the ZED2 camera instead. The proposed method allows to perform object detection by using the ZED Mini directly on Unity without the use of intermediate tools (such as Tensorflow or OpenCV).

To accomplish our purpose, we used the ZED APIs to retrieve the camera images and the depth information. In particular, the video that we receive is from the Left Camera of the ZED, therefore the pose estimation of the object is computed with respect to its frame. In order to perform object detection we imported a pre-trained neural network in the Unity environment via the Barracuda package. Among the possible neural networks we have chosen Tiny Yolo v2 (in ONNX format) to perform detect some common objects.

We tested the code at both high (60) and lower (30) frames per second and we had good results in the second case. In particular we were able to detect correctly a bottle, a chair and people. The whole code is available on Github and it can be downloaded and used by other ZED users [23].

As future work, the whole project could be implemented by using the GPU to retrieve the images, thus obtaining a noticeable reduction of the computational time. Moreover, one or more different neural networks could be added, by also giving the users the possibility of choosing the desired neural network dependently on their needs.

References

[1] M. N. M. N. S. K. Paul, M. T. Chowdhury, object Detection and Pose Estimation from RGB and Depth Data for Real-time, Adaptive Robotic Grasping.
[2] W. X. F. P. C. J. L. X. Liu L., Ouyang W., "Deep learning for generic object detection: A survey," 2020.
[3] W. S. Z. C. C. ZhiguangCao, Tingbo Liao, "Detecting the shuttlecock for a badminton robot: A yolo based approach," 2021.
[4] 2020, technical Specifications ZED-Mini. [Online]. Available: https://www.generationrobots.com/media/zed-mini-camera-datasheet.pdf
[5] 2021, introduction of ZEDMini by SteereoLabs. [Online]. Available: https://www.stereolabs.com/docs/
[6] 2020, depth Sensing Overview. [Online]. Available: https://www.stereolabs.com/docs/depth-sensing/
[7] 2020, getting Started with Unity and ZED. [Online]. Available: https://www.stereolabs.com/docs/unity/
[8] "What is unity?" 01 2020. [Online]. Available: https://www.androidauthority.com/what-is-unity-1131558/
[9] 2020, introduction to Barracuda. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/index.html
[10] 2018, tiny Yolo v2 Overview. [Online]. Available: https://gallery.azure.ai/Model/Tiny-YOLOv2
[11] 2019, yOLO, YOLOv2 and YOLOv3: All You want to know. [Online]. Available: https://amrokamal-47691.medium.com/yolo-yolov2-and-yolov3-all-you-want-to-know-7e3e92dc4899
[12] "Real-time object detection with yolo," 2021. [Online]. Available: https://machinethink.net/blog/object-detection-with-yolo/
[13] 2020, object Detection feature on Unity with ZED. [Online]. Available: https://www.stereolabs.com/docs/unity/object-detection/
[14] 2020, the reason why the interaction between ZEDMini and Unity is not available. [Online]. Available: https://github.com/stereolabs/zed-examples/issues/286
[15] 2019, how to use Pre-trained TensorFlow/Keras models with Unity ML-agents. [Online]. Available: tinyurl.com/db5xwchh
[16] 2020, getting Start with Barracuda. [Online]. Available: https://docs.unity3d.com/Packages/com.unity.barracuda@1.0/manual/GettingStarted.html
[17] 2020, how to use ONNX computer vision models in Unity. [Online]. Available: https://classifai.net/blog/tensorflow-onnx-unity/
[18] 2021, tutorial: Detect objects using ONNX in ML.NET. [Online]. Available: https://docs.microsoft.com/en-us/dotnet/machine-learning/tutorials/object-detection-onnx
[19] 2020, github "ZEDexamples-Tutorial Depth Sensing". [Online]. Available: https://github.com/stereolabs/zed-examples/tree/master/tutorials/
[20] 2020, getting Depth Data. [Online]. Available: https://www.stereolabs.com/docs/depth-sensing/using-depth/#getting-depth-data
[21] 2020, getting Point Cloud Data. [Online]. Available: https://www.stereolabs.com/docs/depth-sensing/using-depth/#getting-point-cloud-data
[22] 2020, basic Concept for the visualization of the ZED in Unity. [Online]. Available: https://www.stereolabs.com/docs/unity/basic-concepts/
[23] "Github repository of our project," 2021. [Online]. Available: https://github.com/sararom15/ObjectDetection-ZEDMini-TinyYolov2-Unity