

Group 3

Adir, #667039

Murat, #642213

Nick, #672542

Computer vision based response time measurement solution

Phase I

# Introduction

**Background**

Ludus Alliance is a sports science technology company based in Israel. They created an interactive event platform for offline and online sports events. The platform connects sports federations, clubs, athletes and provides infrastructure for the meta-verse.

Ludus uses a smart combat paddle as the technological tool to carry out the contests and worldwide cross platform tournaments.

**Problem analysis**

The goal is to replace the current method of measuring the response time for hitting the paddle, which involves using a physical sensor such as an accelerometer, with a computer vision solution. With this new approach, users will be able to capture video of the paddle being hit after a trigger has been lit, and receive the response time as a result. That solution will eliminate the need for implementing the sensor in the paddle, resulting in costs reducing, and better crowd reach.

**Direction**

To address this issue, we plan to utilize a computer vision approach to detect and track two objects: the paddle and the trigger. The trigger will be used to mark the start time of the clock, while the paddle will be continuously tracked throughout the video. We will define a hit as the movement of the paddle above a specific pixel threshold between two consecutive frames. The response time will be calculated as the difference in frames between the trigger being set and the paddle being hit. This value will be multiplied by the video's frame rate to obtain the final result.

# 1. Dataset

The model dataset comprises 323 images capturing the paddle object from various angles within the same indoor lighting environment. We also added 105 images from 10 videos, consisting of key frames we extracted for the analysis. All images are annotated, containing one or two labels, representing the paddle and the trigger (light) being set on. To enhance the dataset, we employed data augmentation techniques, such as horizontal and vertical flips. As a result, the dataset has now expanded to 525 images for training, 57 for validation, and 41 for testing. We split the dataset with a ratio of 70%, 18%, and 12% for training, validation, and testing, respectively.

In this case, using data augmentation in the form of horizontal and vertical flips can help in creating more variations of the paddle and trigger objects, which can improve the model's ability to recognize these objects under different orientations and angles. This is important because in real-world scenarios, the paddle and trigger can be seen from different perspectives, and having a model that can recognize these objects from different angles can be crucial to its overall accuracy.

Horizontal and vertical flips are useful transformations to apply in this case because they can create new samples that are similar to the original data but with different orientations. Horizontal flips can help in creating new samples that are similar to the original data but with a mirror image along the horizontal axis. Vertical flips can help in creating new samples that are similar to the original data but with a mirror image along the vertical axis. By applying these transformations, we can create new samples that are representative of the original data but with different orientations, which can help in improving the model's ability to recognize the paddle and trigger objects.

The dataset can be publicly free-accessed online:

Roboflow: https://app.roboflow.com/inh/trigger-0ubzh/5

GitHub : https://github.com/Adir667/ludus-res-cv

# 2. Model

**Framework reasoning - YoloV5**

For this project we chose to work with the object detection model named YOLOv5. We had several reasons why we chose for this model. First of all YOLOv5 is known for its high accuracy and its speed. For this project it is important that the results come quickly as possible without the fear that the results will not be correct. Other reasons where the fact that it is open source, it has an active community and it has a pretty good documentation.

YOLOv5 is a popular object detection algorithm that has shown impressive results in various computer vision tasks, including object detection, image segmentation, and tracking. One of the main advantages of YOLOv5 is its speed and accuracy, making it a great choice for real-time applications. YOLOv5 is also relatively easy to use and can be fine-tuned on custom datasets with minimal effort. It also supports a wide range of pre-trained models and can be used with various architectures and backbones, making it versatile and flexible for different use cases. Additionally, YOLOv5 has a large and active community, which provides support and resources for beginners and experts alike. For these reasons, YOLOv5 is a suitable choice for object detection in the context of measuring the response time for hitting a paddle using computer vision solutions.

**Programming language - Python**

Python is a good fit for working with YOLOv5 because it is a popular programming language for machine learning and computer vision. It has a large number of libraries and frameworks available that can be used for image processing, data analysis, and deep learning. Additionally, Python has a simple and intuitive syntax, which makes it easy to write and read code, and it also provides a wide range of debugging and profiling tools. Furthermore, YOLOv5 is built using the PyTorch deep learning framework, which is widely used in the Python community. PyTorch is highly optimized for GPU-accelerated computing, making it ideal for training large-scale neural networks such as YOLOv5. Finally, Python has a large and active community, which means that there is a lot of support and resources available for developers working with YOLOv5 in Python.

**The** **algorithm**

Yolov5 is an object detection algorithm that can be used to find and recognize objects in images or videos. It works by breaking down the input image or video into a grid of cells, and then predicting the bounding boxes (coordinates) of any objects that are present within each cell.

These bounding boxes are predicted using a neural network, which has been trained on a large dataset of labeled images. During training, the network learns to recognize common features of objects, such as their shape, color, and texture. Once trained, the network can be used to predict the presence and location of objects in new, unseen images.

Yolov5 is based on a single-shot detector (SSD) approach, which uses a single convolutional neural network (CNN) to perform object detection. This network is composed of several layers that perform feature extraction, object classification, and object localization. The feature extraction layers use a series of convolutional filters to extract high-level features from the input image. These features are then fed to the object classification and localization layers, which predict the class and location of objects in the image.

The classification layer predicts the probability of each object belonging to a particular class. This is done using a softmax function, which assigns a probability value to each class. The localization layer predicts the bounding box coordinates of each object. These coordinates are represented as four values: the x and y coordinates of the top-left corner of the bounding box, and the width and height of the box.

To train the Yolov5 network, a loss function is defined that penalizes incorrect predictions of both the class and location of objects. This loss function is based on the mean squared error (mse) between the predicted and ground truth bounding box coordinates, and the cross-entropy loss between the predicted and ground truth class probabilities.

During training, the Yolov5 network is optimized using backpropagation, which adjusts the weights of the network to minimize the loss function. Once the network is trained, it can be used to perform object detection on new images by feeding the image through the network and using the predicted bounding boxes and class probabilities to identify objects in the image.

Stages :

* Preprocessing: The input image is resized to a fixed size, and the pixel values are normalized to be between 0 and 1.
* Backbone network: The image is passed through a backbone network, which is typically a convolutional neural network (CNN) that extracts features from the image. In YOLOv5, the backbone network is based on the CSPNet architecture.
* Neck network: The features extracted by the backbone network are further processed by a neck network, which typically consists of convolutional and pooling layers. In YOLOv5, the neck network is based on the SPP architecture.
* Head network: The output of the neck network is fed into the head network, which performs the final prediction of bounding boxes and class probabilities. The head network consists of several convolutional layers, and uses anchor boxes to predict the size and location of objects.
* Postprocessing: The output of the head network is processed to remove duplicate detections and filter out low-confidence predictions. Non-maximum suppression is then applied to select the best bounding boxes for each object.

Sources: <https://www.v7labs.com/blog/yolo-object-detection>

https://docs.ultralytics.com/yolov5/architecture/

**Performance – Graphs and metrics**

Confusion matrix – A confusion matrix is a table used to evaluate the performance of a classification model. It shows the number of correct and incorrect predictions for each class in the dataset. The matrix is usually represented as a square where the rows represent the actual or true labels and the columns represent the predicted labels. The main diagonal of the matrix shows the number of correct predictions, while the off-diagonal elements show the number of incorrect predictions. The confusion matrix is a powerful tool for evaluating the performance of machine learning models and can be used to compare different models and tune their parameters to achieve better performance.

Chart

Description automatically generated with low confidence

The confusion matrix shows almost a perfect score, where the paddle is 100% correctly labeled, where the is a slight, insignificant maybe, miss recognitions of the trigger. We could expect some confusion with the trigger since the trigger is very small, reflective and harder to detect.

PR curve – The precision-recall (PR) curve is a visual representation of the performance of a binary classification model. It is created by plotting the precision (y-axis) against the recall (x-axis) at different threshold values used for the classification decision. Precision is the ratio of true positives to the total number of positive predictions, while recall is the ratio of true positives to the total number of actual positives. The PR curve shows how well the model is able to tradeoff between precision and recall. A perfect classifier would have a PR curve that starts at (0,1) and goes to (1,1), indicating that it achieves 100% precision at 100% recall.

Chart

Description automatically generated

F1 – F1 graph is a visualization of the F1 score over time during the training process of a machine learning model. The F1 score is a metric that combines precision and recall, two metrics that are commonly used to evaluate the performance of classification models. The F1 score ranges from 0 to 1, with higher values indicating better performance.

Chart

Description automatically generated

Average precision - When training the model, we have selected epoch value of 1000, hoping to get to the optimal results while letting the training process iterate over and over. By using the default patience value (how long should the model wait for better results, as results can get a little worse before becoming good again) of 10%, we got a smart resulted in a smart stop for us at 191 epochs. As seen in the print, the mAP0.5 has extremely great results, all above 0.96.

Graphical user interface

Description automatically generated with medium confidence

The mean Average Precision (mAP) is a popular evaluation metric for object detection models. It is the average of the precision values at different recall levels, where precision is the fraction of correct detections among the total number of detections made, and recall is the fraction of correct detections among the total number of ground-truth objects.

mAP\_0.5 is a variation of mAP that considers only detections with an Intersection over Union (IoU) score greater than or equal to 0.5 as correct. IoU is a measure of overlap between the predicted bounding box and the ground-truth bounding box, and is defined as the ratio of the area of intersection to the area of union between the two bounding boxes.

In other words, mAP\_0.5 calculates the average precision of the model when it is considered correct if it has correctly identified an object with an IoU of at least 0.5. This metric is commonly used in object detection tasks because it balances the trade-off between precision and recall, and provides a good measure of how well the model is performing in detecting objects.

Metrics in graphs:

Graphical user interface, chart, application

Description automatically generated

The two lines here represent the optimal epoch value of 191, (the orange curve), and the stoppage of the training at 291. The model was not about to perform better after the 191 threshold. The loss function is already in value so close to 0, that it has no effect as for improvement.

Graphical user interface

Description automatically generated

In total, all graphs and metric show very high accuracy results of performance for the model (~0.95).

The model training file can be accessed on GitHub as a Jupiter notebook file:

https://github.com/Adir667/ludus-res-cv/blob/main/Model\_training\_with\_YOLOv5.ipynb

Sources : https://blog.roboflow.com/mean-average-precision/

<https://github.com/ultralytics/yolov5>

# 3. TFGD

**Architecture**

Our technical and functional design includes an API that enables interaction with the developed model. The client will be responsible for managing users and the interaction interface, and is required to authenticate before making any API calls. To authenticate, the client will be given an API key that enables access to the model for response time calculation based on received videos.

**Functional design**

User case diagram

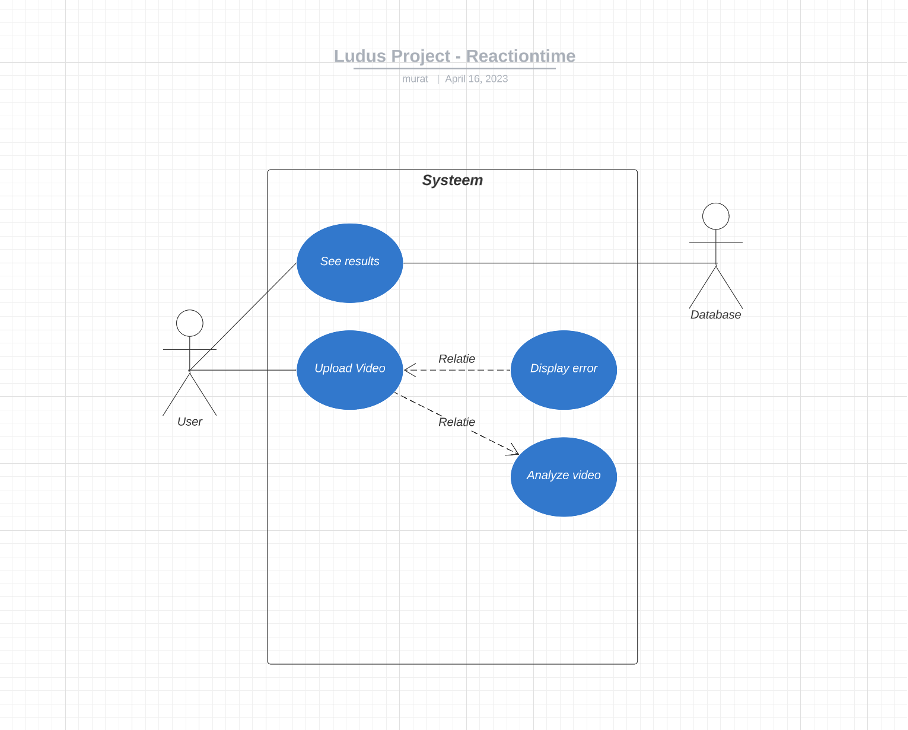
In this use case diagram, the interaction between the user and the system is depicted. The use cases are:

User: the user of the system who records a video and submits it to the system.

System: the system that analyzes the video and calculates the user's reaction speed.

Reaction speed: the use case where the system calculates the user's reaction speed based on the recognized paddle.

Processing return: the use case where the system sends the processed data back to the user, including the reaction speed.



We want to keep the final product as simple as possible with the best result possible. The idea is that the user can make a video on his own device and upload this on our API. The model will analyze the video and calculate the results, if the video meets the requirements. After that the video will be saved in a database so the user can see his previous results and Ludus could use it to analyze the videos as well.

|  |  |
| --- | --- |
| Video analysis | |
| Objective | User uploads a video for response time analysis |
| Preconditions | User is authorized to contact API with right key  Format of the video is .mp4 or .mov |
| Step by step plan | 1. User selects a video to upload  2. User confirm a video to upload  3. Mobile application calls API to analyze the video  4. Model is analyzing the results  5. Result of the response time is sent back to the application  6. Application displays the analysis to the user |
| Postconditions | Video is saved in the company’s dataset  Response time results are saved in a db |

Activity diagram

Diagram

Description automatically generated

**Technical design**

Class diagram

Diagram

Description automatically generated

The flowing classes will be used to program the software for response time detection. The aim is to get the ‘status’ properties of each frame in the video stream and calculate frame gaps when applicable for the time calculation and response evaluation.

API

The way to interact with the system is with API calls to the model for analysis and predictions. The api swagger documentation are publicly accessible here: https://app.swaggerhub.com/apis/Adir667/Ludus-res/3.0.0

# Appendices

**Individual contribution**

**Name: Murat Celem**

1. **Dataset –Model – TFGD -**

During the first three weeks of our project, we worked individually to gain as much knowledge as possible about AI, as none of us had prior experience with it. Personally, I tried and tested various models, including R-CNN and YOLO.

Next, we took many photos from different angles to see which angle would work best. We started from a lower angle but had trouble detecting the lights because they were too small from below. However, this angle would work best because you can see the "hit" well and be more precise. Then we found out that it would work best from the side.

I trained various models in YOLOV5 using different datasets. In the first version, it only recognized the paddle. Then, I trained it on the different sides of the light sensors, but it often got confused. It couldn't distinguish well whether the light was on or off and whether it was left or right. In the last model, I trained it only to recognize green or blue light and the paddle. It indicates that it is active for green light and inactive for blue light.

Then I tried to write code in Python to recognize the paddle. It worked, but not very well, as seen in previous consultations. The video often froze and was very slow. That is why we continued to develop our project using Adir's code.

Furthermore, I worked on the model with the use case diagram within this document. The use case illustrates how we plan to make the application work with the user. The idea is that the user can upload a video through our API, which then analyzes the video. If the analysis is successful, the data is stored in the database and displayed to the user.

**Name: Nick Ryan Rajanayagam**

We needed to create a dataset of images and videos that would be used to train our model. To do this I used my Iphone 14 Pro to capture photos of the paddle from various angles, making sure to cover all sides of the paddle. In addition to capturing still images, we also recorded videos of the paddle being hit and moved around.

Once all the photos were captured, I used makesense.ai to manually label each image so that it could be read by the Yolov5 algorithm. I marked the paddle and the light, so that the alogrithm can learn to detect and track it accurately. After labeling all the photos, I divided them into two groups: one for training and one for validation. The trainng set is used to train the model and the validation set is used to evaluate the performance of the model during training.

As I mentioned earlier, I used Yolov5 for object detection. Durting the training of the model, we encountered some issues where the algorithm would detect object in the background as paddles or it would not detect the lights on the paddle. While I was able to reduce these errors trough accurate labbeling of the images, the performance was still not as good as we had hoped. That is why we decided to use the labled dataset created by Murat using Roboflow. We proceeded to train our model using this dataset.

**Name: Adir Ben Ishay**

* Dataset - After receiving the dataset containing pictures and videos from an iPhone device of my team member, I uploaded them to Roboflow for initial processing and annotations. All images were annotated with either a paddle, trigger, or both. However, the trigger class did not have enough pictures, so I added more pictures to improve its detection. To ensure object detection from all angles, I performed data augmentation using horizontal and vertical flip techniques. I mirrored the paddle and trigger to improve their detection from multiple angles. Roboflow framework was used for data augmentation, which has the advantage and flexibility to export data and annotations to various other platforms and frameworks.
* Model - Initially, I conducted research to determine the available tools and frameworks. I chose YoloV5 for its accessibility, ease of use, and fast processing. I followed the instructions in the Yolov5 custom detection instruction file for model training. To alleviate the CPU burden, Google Colab resources were used for training. The model was trained multiple times, and optimization was achieved by evaluating the performance graphs as described in section 2. The model was tested using code and video analysis for response time calculation, and the results were promising. During the consultancy sessions, we were able to show good prediction results and high precision.
* TFGD - Currently, the architecture is designed to allow clients to interact with the system and analyze videos for response times after the video has been taken (the implementation of live video stream is not yet discussed in this document). Assuming that the client communicates with us through an API (from a mobile app or other user interface), the proper solution for this architecture is to build an API. I designed the API using the Swagger framework, which is publicly available. The API has two basic functions, POST for analysis and GET for videos that have already been analyzed. When posting to the API, the user must include the user\_id, club\_id, app/interface\_id, and of course, the video file in supported formats such as mp4 or mov. The response contains an array of response-time detected by the model. Additionally, I designed the activity and class diagrams.