Implementation of an Interactive Human vs Robot "Water Pong" Game

Clément Detry, Loris Podevyn, Ivan Poliakov, Bunyamin Thijssen
Department of Data Science and Artificial Intelligence
Maastricht University
Maastricht, The Netherlands

***Abstract*—This article describes an interactive game involving a human and a throwing robot called "Water Pong". The goal is to throw a ball into several cups disposed in a triangular form. The robot is defined by three degrees of freedom, which allows it to throw towards a target with a parabolic trajectory. Cups are located using Computer Vision, giving the robot the ability to see the target it needs to hit. This vision is also used to track the projectile launched by the human and the robot to define a successful shot.**

***Index Terms*—AI, Interactive Game, Computer Vision, Ball Throwing**

I. Introduction

In recent years, the idea of using robotics to adapt existing interactive games has become less science fiction and more of a reality. Whether it is for training purposes in sports [1], in rehabilitation [2] or purely in entertainment as proposed through this paper, many fields have been positively affected by the contribution of robotics. Nowadays, many games have already been adapted into a kind of human against robot format and are increasingly available and affordable on the market.

The goal in this research is to develop the modern day party game of "Water Pong", in which the user can interact and play against a robotic arm. As the robotic arm mimics a player throwing, there exist several complexities to control its movements: recognizing targets with a fixed-positioned camera and calculating the optimal trajectory and motor configurations to perform the shot. This allows us to research in Computer Vision to develop a way to detect the cups, and follow the score of the game by tracking the ball in real-time to see if either the player or the robotic arm hit one target. Other research is being conducted on building trajectories in order to make the robot accurate and robust to various cup positions. Along with research, we would like to implement an interactive aspect of the game so that the robot can track the score, the rules, and collect data about the person it is playing against.

In section I-A, our work is compared to the current state-of-the-art. Section II provides an insight of the hardware we used, and the environment of the robotic setup. Section III explains the algorithms developed. Section IV contains the description of the experiments, and their results are given in section V. A discussion and some interpretations of the results are included in section VI. The section VII concludes our work and section VIII gives a brief perspective on possible future work to improve on our developments.

This report was prepared in partial fulfilment of the requirements for the course KEN3300 Project 3-1 at the Department of Data Science and Artificial Intelligence, Maastricht University. Supervisors: Rico Möckel, Lucas Dahl. Examinators: Rico Möckel and Alexia Briassouli.

A. Related Work

Playing an adapted version of "Water Pong" has already been done before [3] with a robot called Versaball developed by Empire Robotics [4]. However, we are the first to use EDMO servo motor [5] technology to perform the robot throws for this kind of game, as Versaball uses air compression and a gripper [6]. For the Computer Vision part, we took different elements from several projects. To perform cup detection, the most advanced technique that met our needs was to use Hough Circle Transform [7]. In order to track the ball in real-time, and any existing QR-codes on the image, the YOLO algorithm was chosen [8]. Specifically, the fifth and last updated version of YOLO was employed [9]. After developing the detection of QR-codes, which are our reference objects, as well as the detection of cups, we needed a way to measure the actual distances between the detected objects. We were inspired by an article [10] to implement this task in this way.

II. Hardware and Environment Setup

A. Microcontroller

The microcontroller being used is an Adafruit Feather M0 board. The script for package execution is written in C and has to be submitted only once to the microcontroller via Arduino. All the motors are connected to the microcontroller, which allows for simultaneous movement of all the components. The package has the form of $a_0, b_0, c_0, a_1, b_1, c_1, ...$ where each tuple of 3 values represents the motor number, target angle and delay respectively. The data package is transferred via USB cable causing data transfer overhead that interrupts the motor movement, hence, the first tuple is always filled with heap data and required for the shot motor movement starts only after the overhead is avoided.

B. Motors

The motors in this project are EDMO servo motors [5], which just like the spoons and the camera were kindly provided to us by DKE Swarmlab at Maastricht University [11]. All the motors are depicted in Figure 1 as they are

positioned in the setup. The bottom motor is responsible for rotation of the setup and, therefore, the direction of the shot. The top 2 motors are responsible for throwing the ball. The throw distance is controlled by the delay on the top motor (for example, if the top motor starts its rotation 0.1 seconds after the middle motor started its rotation, the throw distance is smaller than if the motors were moving simultaneously). The top motor is coloured black, which indicates it is lighter and weaker than the red motors. Thus, the black motor is mainly used for throwing distance control and increased momentum, while the middle red motor is the carrying power of the throw.

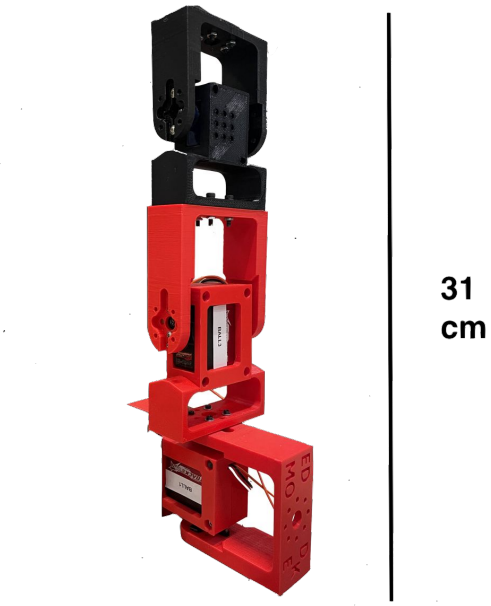


Fig. 1: Motors

C. Spoon

Spoon is attached to the top motor and it carries the ball. Since the initial maximum throwing distance was unsatisfactory, several longer spoons were developed by our colleagues in SwarmLab. Longer spoons provide bigger momentum, however, they also make the setup more shaky. This was attempted to be solved with a more solid material and structure of the spoon (third spoon in Figure 2). However, the motors could not deal with such a weight, so it was decided to move back to a lighter material but keep the spoon structure (fourth spoon in Figure 2) which appeared to be the best option.

D. Ball

The ball chosen for this project is a navy blue fluffy ball of 3.5 cm in diameter that weighs about 1g as we can see in Figure 3. The main advantage of this ball is its consistency which allows the robot to throw the ball farther than a classic ping-pong ball.

Fig. 2: Spoons

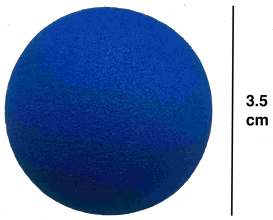


Fig. 3: Ball

E. Cups

The cups used were classic red plastic cups, 12 cm high and 10 cm wide as shown in Figure 4. The issue with this type of cup is that the impact with the ball can make them fall, because they are very light. To solve this problem, we decided to add weight with small stones, but the ball would now bounce out of the cups when the player scored it. The final solution used was to add white sand inside them to cushion the ball every time a cup is hit. This also allows the ball to be elevated in the cups so that the camera can continue to detect it anywhere on the board within the targets. The choice of sand rather over liquid was made because of the material and density of the ball changes when it is covered in water, but we want to avoid influencing its weight during the game.

Fig. 4: Cup

F. Camera

To detect the game board and the different objects placed on it, a camera is positioned on an overhead arch at 97 cm height in order to be able to capture the whole game with a wide enough opening angle. Two different types of cameras were tested. The first one was a Creative Live Cam Sync HD, filming at 720p resolution with a frame rate of 30fps and without autofocus. The quality of this camera was not good enough when it came to tracking the ball during throws, since it detected too much blur. To solve this issue, a second camera was considered, a Logitech C922 Pro (Figure 5). The main difference of this camera is that it can capture at 60fps and it can use autofocus, which reduces the blur caused by the rapid movement of the ball. The camera is directly connected by USB to the computer since it is the component responsible for operating the entire Computer Vision system.

Fig. 5: Logitech C922 Pro

G. Environment and Setup

The environment in which the tests of the game "Water Pong" were carried out, is in a large closed room with good, adjustable lighting. No prior knowledge about the environment is given to the robot before each execution. The final game setup is a rectangular board made of a 133 by 43 cm piece of wood covered by a large green grid paper spread on top of it. The overhead arch is also made of wood and is not fixed as it can be moved forward and backward. The camera is attached to it using a loop fastener to maintain it as stable as possible. Each side of the board contains 6 cups placed in a pyramid shape at the beginning of each game. The last element added to the board is 2 QR-codes to be used as reference object to calculate the position of the cups in real-time on the grid. Figure 6 represents what the fully assembled setup looks like.

III. Methodologies

A. Look-up Table

To allow for precise shots, we built a large look-up table that contains motors configuration (bottom motor angle and top motor delay) per target. The targets can be seen in Figure 7. The entries can be visually split into vertical columns. First, the delay is determined based on the delay for the entry from the look-up table that is closest to the real target. This is due to the fact that it is hard to establish a relationship between the delay and the throwing distance, so the delay value from the look-up table is taken as it is(see Figure 8 and

Fig. 6: Assembled setup

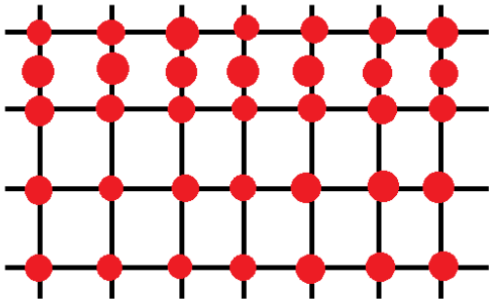


Fig. 7: Lookup table

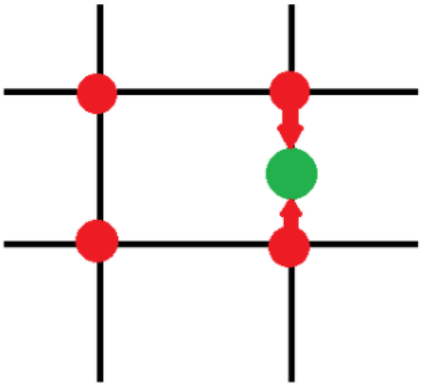


Fig. 8: Lookup table 1

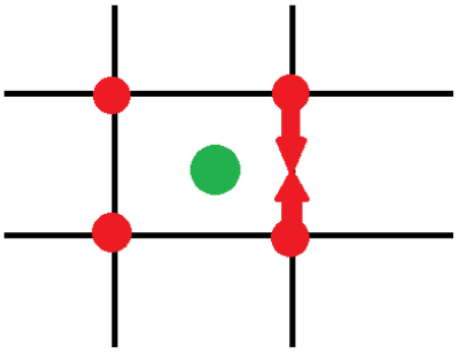


Fig. 9: Lookup table 2

Figure 9). Then, the second closest entry from the look-up table is chosen and the bottom motor angle is interpolated between the two suggested angles from the look-up table. As it turns out later in the experiments, all of the missed shots hit the edge of the cup, which can be resolved by adjusting the delay. Therefore, in the actual game we added the feature that the robot starts adjusting the delay by ± 10 if it cannot hit the target (so at first -10, then +10, then -20, etc).

B. Hough Circle Transform

It is essential for the robot to detect the cups and know their positions in order to shoot the ball into them. The Hough Circle Transform (HCT) technique is a famous way to find the center coordinates of circular shapes in an image [12]. Since we have a fairly large contrast between the cups (white) and the background (green), HCT fully meets our needs. A circle with radius r and center (a, b) can be described with the parametric equations:

$$x = a + r * \cos(\theta)$$

$$y = b + r * \sin(\theta)$$

If the radius of the search circles on an image is known, the objective is to find coordinates (a, b) of the centers. In Figure 10, each point in the geometric space on the left generates a circle in the parameter space on the right. The circles in the parameter space intersect at (a, b) that corresponds to the center in the geometric space.

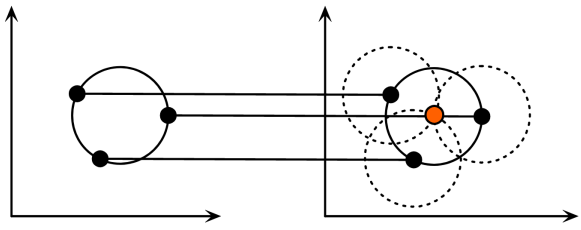


Fig. 10: Parameter space [12]

In order to find multiple circular shapes in a single image, an accumulation matrix is computed by HCT. The accumulation matrix of the parameter space, for example in Figure 11, contains the values of the central coordinates at the peaks. If the z-value in the 3-dimensional space is higher than a certain threshold, the circle is taken into account.

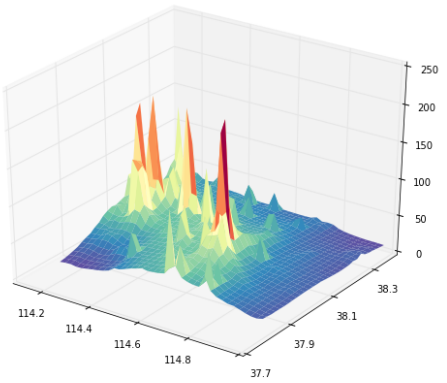


Fig. 11: HCT parameter space accumulation matrix [13]

The HCT algorithm is already implemented within the OpenCV library in python. This is the version we used for our cup detection [7].

C. YOLO

YOLO, acronym for "You Only Look Once", is an object detection algorithm that detects and recognizes items within images in real-time. This approach is appreciated because of its detection speed and accuracy. Recognition in YOLO is performed as a regression task. What makes the algorithm fast is that it only requires a single forward propagation through the network to detect objects in whole images [14]. YOLO implements Convolutional Neural Networks to predict various classes and bounding boxes simultaneously. It splits the images into cells, each of which is responsible for predicting k bounding boxes. As we can observe in the Figure 12, each small square in the grid holds k vectors, in this case 5, of size $(5 + n_c)$ with n_c being the total number of prediction classes [15]. The first 5 elements of the vector are the properties of the bounding box. The center of the box is defined by the coordinate (b_x, b_y), the width and the height by b_w, b_h and p_c is the probability that class c is the bounding box.

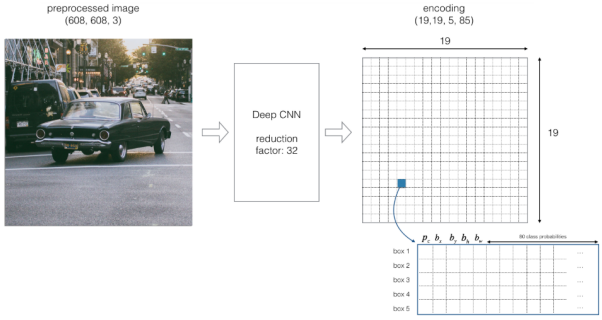


Fig. 12: Bounding box regression [15]

During the single pass of the forward propagation, YOLO calculates the probability that the cell includes a specific class. The corresponding equation is as follows:

$$score_{c,i} = p_c * c_i$$

The class with the highest probability is picked and allocated to that particular grid cell. A similar process occurs for all grid cells in the image. After calculating the above class probabilities, the image may look like this:

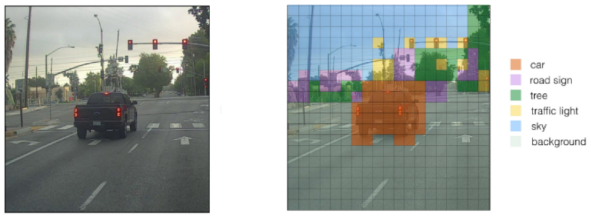


Fig. 13: Image after computing class probabilities [15]

After the prediction of class probabilities, the next step is called the non-max removal, which helps the algorithm to eliminate unneeded anchor boxes. As we can see in the Figure 14, there are many anchor boxes calculated based on class probabilities.

Fig. 14: Anchor boxes image [15]

To solve this issue, non-max suppression function eliminates bounding boxes that are very similar by performing Intersection over Union (IoU) with the one that has the highest class probability among them. Figure 15 gives an overview of how the method works.

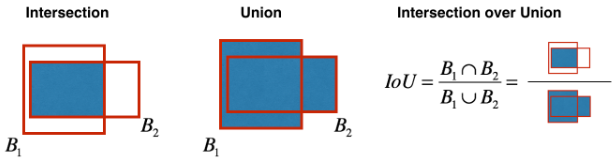


Fig. 15: Intersection over Union operation [15]

YOLO rejects all bounding boxes that are covering the same object whose IoU value exceeds a certain threshold. This process is repeated until all the different bounding boxes are obtained. The Figure 17 gives an example of the non-max suppression method effect.

Fig. 16: Non-max suppression effect [15]

To train a model of YOLO on our custom data, we used the open-source ultralytics version [16] on Google Colaboratory to take advantage of their GPU. The YOLO architecture is composed of 25 layers, mainly standard convolution, 3-dimensional convolution, tensor concatenation and the final detection layer.

1) Custom Dataset

Our custom dataset consists of 388 images of the game board with manually labeled with QR-codes and balls. To train our YOLO model more robustly, we decided to add random noisy objects to the board for each image. The software used to label the images is called "makesense.ai" [17]. It allowed us to export the files containing the labels in the correct format for YOLO. For the training part, the dataset was divided into two different sets. The training set, composed of 273 images ($\approx$ 70%) and the validation set of 115 images ($\approx$ 30%). We decided not to use a test set in order to keep as many images as possible for training purposes.

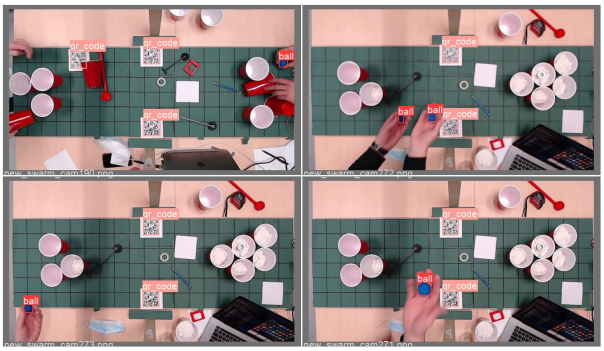


Fig. 17: Custom dataset image examples [15]

The Figure 18 shows some insights about the labels in our custom dataset. The left bar plot shows that there are a larger distribution of QR-code ($\approx$ 650) than ball labels ($\approx$ 350). The density graph on the right is also interesting to observe because it shows the most common locations of both types of labels on a single plot at the same time. As expected, the most frequent spot is around the two QR-codes. Besides that, the other labels appear to be distributed normally.

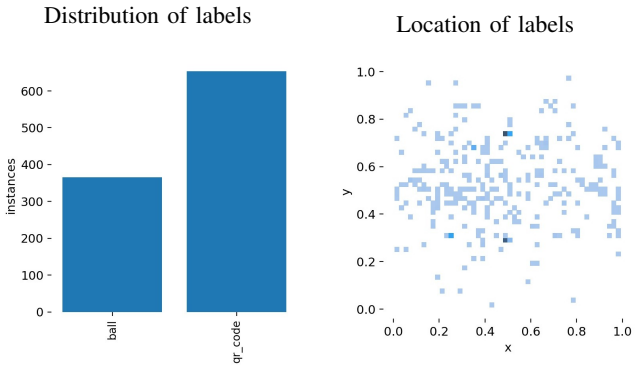


Fig. 18: Label insights

D. Neural Magic

Neural Magic's latest algorithms enable Convolutional Neural Networks such as YOLO to run on commodity CPUs at GPU speeds [18]. To achieve this, it uses sparsification through pruning of the model to reduce the weight's size by 10 times or more. Sparsification is the process of taking a trained deep learning model and removing the redundant information from the overprecise and over-parameterized networks. Neural Magic provides the option to apply different types of recipes for model training, from pruned to pruned quantized as shown in Figure 19. Pruning is the compression method of removing unnecessary weights from a trained model. Quantization refers to the process of approximating a neural network that uses floating-point numbers to a lower bit width numbers.

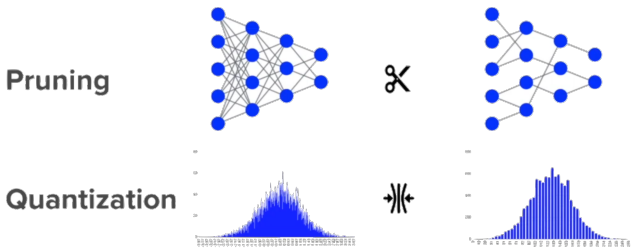


Fig. 19: Pruning and quantization illustration [18]

After training the sparsified model, the DeepSparse engine can be used for cheaper inference deployment. The DeepSparse engine is a CPU runtime that offers breakthrough performance by leveraging the natural sparsity of neural networks to reduce computational requirements and accelerate memory workloads [19]. Figure 20 illustrates a summary with the path we used for the sparsification of our YOLO model.

E. Object Distance Measurement

To measure the distances between objects with SI units, it starts with identifying a reference object. This object should have two important properties. We must know its dimensions in some measurable unit, and we must identify it easily in the image [10]. Our refrence objects are the QR-codes that

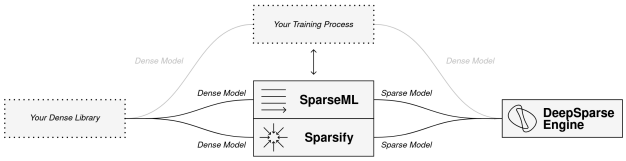


Fig. 20: Summary to sparse acceleration [18]

are part of the playing field. Next, it is sufficient to take the results obtained by YOLO for the locations of the QR-codes and by Hough Circle Transform for the positions of the cups. Thanks to the euclidean distance formula we can calculate the pixel distances between the centers of each cup and the two QR-codes. In order to get them in centimeters, we need a reference object ratio [20] to divide them. The formula to get the distance is the following:

$$ratio = \frac{p}{d} \qquad (1) \qquad dist_{cm} = \frac{dist_{px}}{ratio} \qquad (2)$$

where p is the length of the reference object calculated from YOLO in pixels and d is the length of the reference object measured by hand in centimeters. Figure 21 illustrates a basic example of the object distance measurement method.

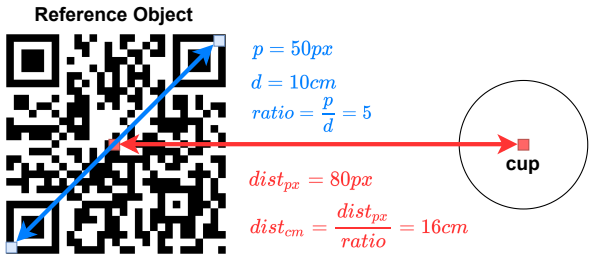


Fig. 21: Distance measurement example

Having distances in centimetres, it was easy to implement a function to compute the ball speed by adding a time parameter. The distance in cm between two detections of the same ball is divided by the time spent between these two detections. After conversion, if the speed is above 10 km/h, it is considered a throw. The direction from which the ball is coming from, according to the QR-code helped us to define when a throw was made either by a human or the robot.

IV. Experiments

A. Throwing Accuracy

Since we are using a look-up table, it is hard to come up with a way to analyze the throwing accuracy of the setup. Each value in the lookup table impacts a very small region on the grid, so there is no general method to assess quality of all entries together. Hence, the experiments mostly focus on exploring the quality of the angle interpolation and additionally provide us with useful insight into the accuracy pattern of the setup. First, we measure accuracy out of 10

shots in the predefined positions in the lookup table that does not require any approximations, to detect possible deviations associated with the motor movement, setup shaking, etc. Then, we make 10 shots into 10 different spots with known delay values and 10 shots into different spots with "in-between" delay values (that is, the target cup is located "between the columns" in the look-up table). As it was mentioned earlier, the delay values are taken from the look-up table as they are, so it is interesting to see how well angle interpolation does together with delay value that is crudely approximated.

B. YOLO Algorithm Detection Accuracy

In order to test the full potential of YOLO on our setup, we decided to tweak one of the main training parameters, the resolution of the input images. We decided to go from 160x90 pixels image to 640x360, which should drastically change the detection efficiency, as the difference in quality is great. After being trained on our custom dataset of 388 images, the YOLO model was able to obtain some decent results. Each model was trained over 200 epochs, which represents about 5 hours of computation. In addition, an early stop condition was implemented to avoid overtraining our models. If no learning improvement is noticeable over the last 100 epochs, the process is stopped and the weights saved. The following measurements will be used to determine the effectiveness of YOLO prediction.

1) Confusion matrix:
A confusion matrix is a specific table that allows the visualization of the performance of a learning model, generally a supervised one. It enables us to derive a variety of measurements that can reveal some important prediction characteristics. Figure 22 shows an example of what this looks like.

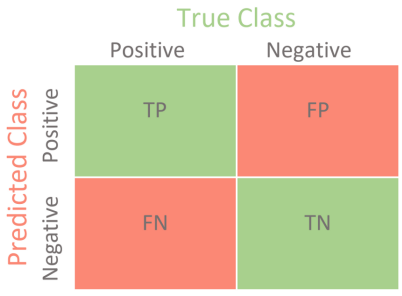


Fig. 22: Confusion matrix example

2) Precision:
Precision determines the ratio between the true positives and all the positives instances. A high value means that our model returns more relevant results than irrelevant ones. Precision is seen as a measure of quality over quantity [21].

$$\text{precision} = \frac{\text{TP}}{\text{TP + FP}}$$

3) Recall:
Recall is the measure of our model correctly classifying true positives [22]. A high value means that our model returns most of the relevant results. Recall is seen as a measure of quantity over quality [21].

$$\text{recall} = \frac{\text{TP}}{\text{TP + FN}}$$

4) mAP@[.5]:
The Mean Average Precision (mAP) is a useful metric for Computer Vision methods such as YOLO [23]. It is calculated using IoU that we already discussed in section III-C, Figure 15. The mAP@[.5] validates the object's prediction on a specific image if the IoU scores above a threshold of 0.5. For example, if the IoU value for a prediction is 0.7, then we classify the prediction as a true positive instance. On the other hand, if IoU is 0.3, we classify it as a false positive. The formula is as follows:

$$mAP = \frac{1}{|classes|} \sum_{c \in classes} \frac{\#TP(c)}{\#TP(c) + \#FP(c)}$$

where TP/FP defines the number of true positive/false positive instances using a IoU threshold of 0.5 [24].

5) mAP@[.5:.95]:
This metric is similar to the mAP@[.5] but deals with different IoU thresholds, from 0.5 to 0.95 with a step size of 0.05. An average among these thresholds is then calculated.

C. Neural Magic Application

For this next experiment, we want to test the benefit of using Neural Magic and the different tools it offers to train a sparsified YOLO model and use a more powerful inference engine. As mentioned in their documentation, this should allow us to have lighter output weights for our model with faster deployment. The task now is to verify whether opting for inference speed rather than better detection performance is a good decision for our setup or not. To address this question, the identical experiment with the same data and environment as the one used for YOLO was performed. The objective is to obtain results that are as consistent as possible.

D. Computer Vision Limitations

After having the object detection model trained and all the different components of the Computer Vision merged together, we wanted to test the possible limitations of it. The goal was to find out when the Computer Vision implementation could break down or be robust after changing the state of the game environment. To determine this, some of the main components of the setup were modified. First of all, the intensity of the lights surrounding the game board was tweaked. Since the lighting in the experimental room was adjustable, the experiment was conducted during the

evening by building a custom scale from 0% to 100%. Then, different amounts of Gaussian noise were applied to the image. Finally, the speed of the thrown ball was gradually increased to determine different speed detection threshold.

E. Framework

The experiments on YOLO and Neural Magic were carried out using the open-source ultralytics project [16]. The deep learning part of this project was performed with PyTorch 1.7.0 using the Google Colab GPU. Concerning the hardware, the experiments were conducted on a laptop composed of a 6-Core Intel i7 with an Intel UHD graphics 630 GPU and 16GB Ram.

V. Results

A. Throwing Accuracy

From the first experiment we measured the accuracy of 70% to 90%. The ball went in the cup 7 times, 2 times it bounced off the cup while being inside the cup. Therefore, technically it can count as hit since the cups had sand instead of water inside them to keep the ball dry.

The remaining two experiments reinforces the conclusion that 90% should be taken as the standard accuracy of the setup as they both generated 90% accuracy as well.

Throughout the experiments all misses that did not involve the ball bouncing off inside the cup turned out to be hits on the edge of the cup. All the missed targets were then successfully hit by the very next shot without post-miss calibration involved. The fact that the only observed opportunity to miss is hitting the edge of the cup suggests that adjusting the delay of the top motor by ± 10 after the miss is an effective method to eventually hit the cup in several attempts.

B. YOLO Algorithm Detection Accuracy

The bar plot in Figure 23 represents the confusion matrices after YOLO training for the different input image resolutions. As we can see, the true positive and false positive values for the custom labels are reported in a normalized form. The two clusters on the left illustrate the percentage of correct labels found by YOLO. Therefore, we can state that higher resolution images lead to a slight increase in performance in detecting ball instances. However, for QR codes, even the lowest image definition can detect them 100%. The two clusters on the right define the difference between ball and QR-code objects that are falsely detected after the model is trained. This clearly proves that on average more balls are detected in the background compared to QR-codes for any image resolution.

The line charts in Table I shows both mAP@[.5] and mAP@[.5:.95] metrics recorded over the training epochs of YOLO. We can observe a small improvement in learning speed with the increase of image resolution. Indeed, the 640x360 parameter appears to start converging in mAP@[.5] after only 18 epochs, while the 160x90 images start converging after 50 epochs. On the other hand, the mAP@[.5 :.95] learning curves are a bit more chaotic from the two highest

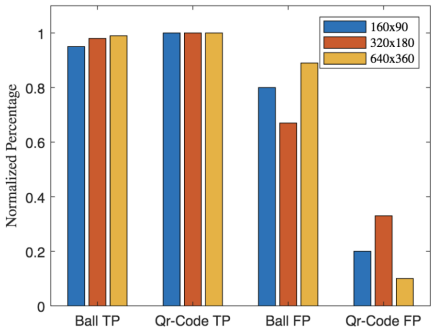
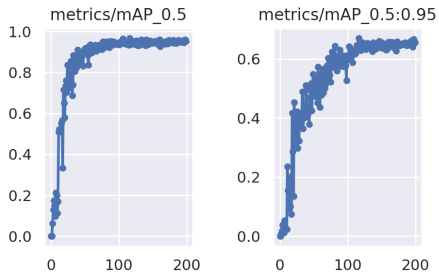
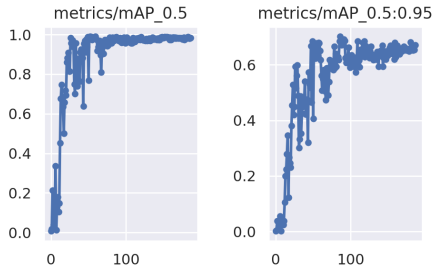
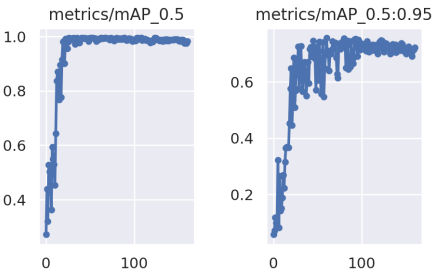


Fig. 23: Confusion matrix bar plot

image definitions but still get slightly better performance than the lowest resolution.

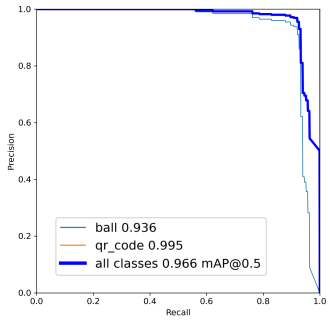
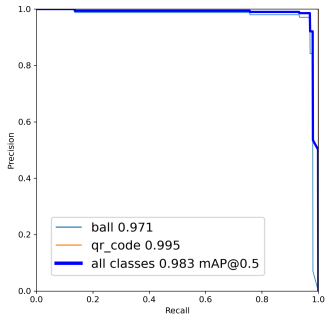
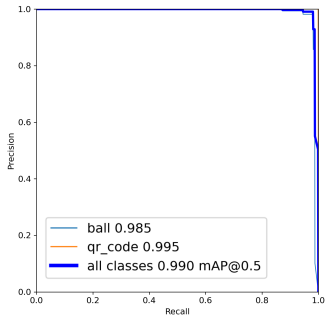
TABLE I: mAP over Epochs

Resolution	Graph
160x90	
320x180	
640x360	

To measure the precision and the recall of the models, Precision-Recall Curve (PRC) are used and represented in Table II. The choice to use the PRC rather than the ROC

curves is due to the balancing of our custom dataset, Figure 18. ROC curves represent the trade-off between the true positive and the false positive rate for a predictive model using different probability thresholds. They are appropriate when the observations are balanced between each class, while the ROC curve is relevant for unbalanced datasets [25]. As we can see in Table II, the three different models perform well because the curve of all classes is close to the upper right corner, which means that we have obtained a model with perfect detection skill. Still, there is a larger difference between the 160x90 model and the 320x180 model with an increase of 0.017 in the area under the curve (corresponding to the mAP@[.5]) for the best values compared to an increase of 0.007 between the 320x180 model and the 640x360 model.

TABLE II: Precision Recall Curve

Resolution	Graph
160x90	
320x180	
640x360	

Regarding the frame rate at which YOLO can process images, we have various ranges for the different resolutions in Figure III. The highest speed, between 13 and 16 fps, is obtained with the lowest image definition. Similarly, the lowest detection speed, between 2 and 3 fps, is achieved with the highest resolution. In addition, the size of the file containing the model weights is quite consistent among the three different groups, at about 14.3 MB.

TABLE III: Frame Rate

Resolution	160x90	320x180	640x360
Weights	14.2 MB	14.3 MB	14.4 MB
fps	13 - 16	6 - 8	2 - 3

C. Neural Magic Application

There is no big difference in detection performance using Neural Magic's sparseml tool to train a sparsified version of YOLO, as we can see in Figure 24. If we compare it to the bar plot 23, the only point to mention is the gap between the ball and the QR-code FPs which is smaller using the model sparsification.

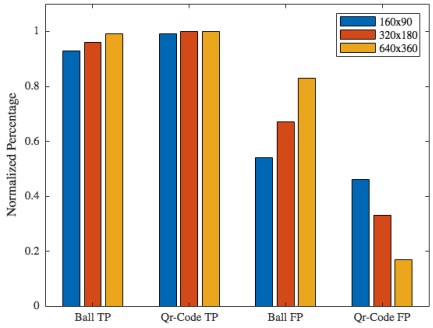


Fig. 24: Sparseml Confusion matrix bar plot

For the mAP results over the epochs (Table IV) and the PRC (Table V), we decided to only keep the graphs of the 320x180 model in the core of the paper since the results have not changed much nor add additional information. The remaining figures are in the appendix, Table XI and XII. As we can observe, Table IV proves that pruning the weights of our model as well as its quantization has not reduced much the detection efficiency of YOLO during learning in terms of mAP. Its learning speed is a bit slower than previously in Table I but it still starts to converge after 85 epochs.

Concerning the Precision-Recall Curve, the difference between the original YOLO and its sparsified version is minor. If we compare the graphs in Table II and V, they are very close to each other. The only point to mention here is a small 0.009 decrease in the area under the curve.

In Table VI, the new fps values using the more powerful Deepsparse inference engine are reported. The highest fps obtained increased significantly compared to that obtained with the original YOLO inference engine (Table III), around 3x faster. Furthermore, the size of the model weight files has been reduced to 7.2 MB, which is a reduction equivalent to 2x the previous size.

TABLE IV: Sparseml mAP over Epochs

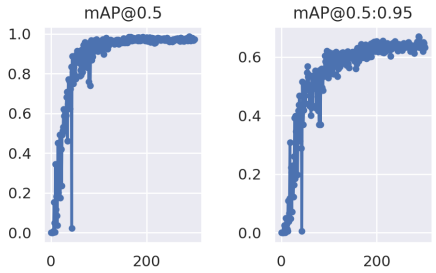
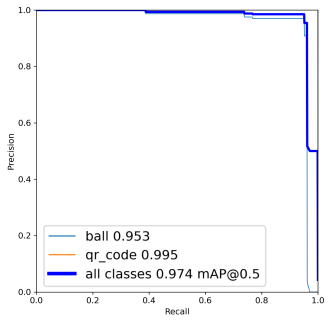
Resolution	Graph
320x180	

TABLE V: Sparseml Precision Recall Curve

Resolution	Graph
320x180	

D. Computer Vision Limitations

1) Lights

Table VII refers to the results obtained with the different light intensities in the experimental room. As a result, the Computer Vision is not able to detect cups or balls without any light in the room. However, as soon as there are any lights, all objects are detected in the frame up to the highest intensity.

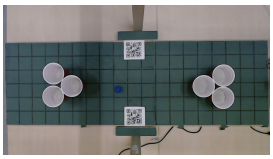
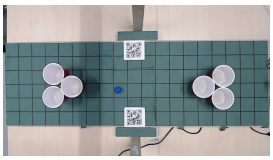
2) Image noise

Table VIII shows the results after applying Gaussian noise to the input image with different mean (μ) and standard deviation (σ) values ranging from 0.001 to 0.1 for both parameters. As the values increases, the detection is less efficient. With a value of 0.001 for the noise distribution parameters, every object on the setup are detected. The two QR codes are predicted with a probability of 92% and the ball 72%. Afterwards, the Gaussian noise parameters are multiplied by 10. In this case, the objects are still detected

TABLE VI: Deepsparse Frame Rate

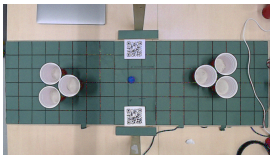
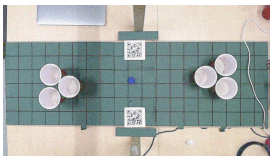
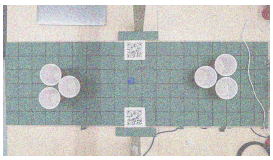
Resolution	160x90	320x180	640x360
Weights	7.2 MB	7.2 MB	7.3 MB
fps	40 - 47	24 - 27	8 - 9

TABLE VII: Lights Intensity

Lights	Image	Detected
0%		No detection in the image
25%		2 QR-codes (87%), 1 ball (68%), 6 cups
100%		2 QR-codes (91%), 1 ball (78%), 6 cups

but with a lower accuracy, about 80% for the QR-codes and 60% for the ball. Finally, with μ and σ equal to 0.1, the computer vision detection appears to be broken because no correct prediction is made and many random false positive instances are predicted over the whole image.

TABLE VIII: Gaussian Noise

Noise	Image	Detected
μ = 0.001 σ = 0.001		2 QR-codes (92%), 1 ball (72%), 6 cups
μ = 0.01 σ = 0.01		2 QR-codes (80%) + $\approx$ 3 FP, 1 ball (60%) + $\approx$ 2 FP, 6 cups
μ = 0.1 σ = 0.1		No good QR-codes + many FP, no good ball + many FP, no cups

3) Ball speed for detection

After throwing the ball several times at different speeds between 25 and 60 km/h, we were able to approximately discern the speed ranges at which the camera had trouble

detecting the ball using our model capable of producing about 45 fps. This led us to construct confidence intervals presented in the table IX which shows the percentage of balls detected below and up to a certain speed. As we can see, below 35 km/h, the ball is always detected, above, the ball can be missed by the computer vision. The maximum speed detected was 67 km/h.

TABLE IX: Ball speed detection

Confidence	**25%**	**50%**	**75%**	**100%**
Ball speed	$\leq 52km/h$	$\leq 45km/h$	$\leq 41km/h$	$\leq 35km/h$

VI. Discussion

Experiments conducted on the Computer Vision have shown multiple interesting features through their results. From section V-B, we learned that QR-codes were slightly easier to detect than balls. This is certainly due to the fact that QR-codes have a fixed square shape compared to the balls, which can have either a circular shape or a sort of long ellipse if the object is moving quickly and then blurred in the dataset. There is also more contrast between the QR-codes and the green background. Regarding the YOLO training, we can say that after 320x180 resolution images, we do not observe a significant increase in mAP, precision and recall. This might be a good trade-off to stick to this picture definition. In this section, we also noted the strong impact of using higher image resolution for the real-time detection speed. The interpretation of this effect is simply the big difference in the total amount of pixels YOLO needs to handle between the three image resolutions analyzed.

160x90 = 16200 pixels

320x180 = 64800 pixels

640x360 = 230400 pixels

In section V-C, we found that the use of pruning and quantization of the YOLO model has not decreased the prediction performance obtained previously. In addition, the use of the Deepsparse inference engine has significantly improved the detection rate by up to 3 times. The fact that the weights files are twice as light as before played a role for this, thanks to a less expensive model deployment.

The robustness results of the Computer Vision, tested in section V-D, revealed a few interesting elements to mention. Changing the lights in the room did not interfere much with the detection as long as they were not turned off. This is because YOLO is known to be very solid and the green background helps it by not reflecting the lights. Concerning the noise in the image, it starts breaking the Computer Vision after adding a Gaussian noise with a mean and a standard deviation of 0.01. This is explained by the fact that noise interferes a lot with the pixel value in the image and YOLO is not trained to have that much pixel variation in the input frames. Regarding the ball speed, our experiments show that the frame speed and the input image resolutions are enough to always detect ball thrown below 35 km/h. Since the average throwing speed of the robot is about 19 km/h and about 25 km/h for the human, we can say that the ball speed detection is efficient enough to not miss a single shot.

VII. Conclusion

In this research, we developed a "water pong" game that allows human to play interactively with a robotic arm. The main challenge was to develop a robust Computer Vision capable of detecting several key objects such as the target cups, the ball, and the QR-code reference element, along with a means of calculating the optimal motor configuration to shoot the targets. We relied on the well known YOLO algorithm for the real-time object detection as well as Hough Circle Transform to identify the cups locations. The option to go for an extended lookup table with additional heuristics to obtain the correct motor settings was employed. The communication between the cups positions and the lookup table made it possible for the robot to play the game targeting and shooting in most of its opponent's cups with a global accuracy 90%. To the best of our knowledge, this is the first working demonstration of a "water pong" game using EDMO servo motors for the robotic setup.

VIII. Future Work

An improvement over the current state of the project would be to implement a robust way to calibrate the lookup table using the Computer Vision. To achieve this, we would need a better equipment, both camera and computer in order to measure less blur when the ball is moving fast and increase the detection speed. This new feature would improve the throwing accuracy by estimating when the ball is overshooting or undershooting the targets.

References

[1] C. Opfer. *Robot Pingpong Coach Helps Players Up Their Table Tennis Game*. 2017. URL: https://science.howstuffworks.com/table-tennis-pingpong-robot-coach-forpheus.htm.

[2] R. Gassert and V. Dietz. *Rehabilitation robots for the treatment of sensorimotor deficits: a neurophysiological perspective*. 2018. URL: https://jneuroengrehab.biomedcentral.com/articles/10.1186/s12984-018-0383-x.

[3] A. LaFrance. *Meet the Robot Champ of Beer Pong*. 2015. URL: https://www.theatlantic.com/technology/archive/2015/01/meet-the-robot-champion-of-beer-pong/384285/.

[4] Empire Robotics. *About us*. 2014. URL: http://www.empirerobotics.com/about/.

[5] R. Möckel, L. Dahl, and S. M. Christopher. *Interdisciplinary Teaching with the Versatile Low-Cost Modular Robotic Platform EDMO*. 2019. URL: https://link.springer.com/chapter/10.1007/978-3-030-18141-3_11.

[6] J. Flaherty. *A Beanbag Robot Hand That Works Insanely Well*. 2018. URL: https://www.wired.com/2014/01/empire-robotics-jamming-robot/.

[7] A. Rosebrock. *Detecting Circles in Images using OpenCV and Hough Circles*. 2014. URL: https://www.pyimagesearch.com/2014/07/21/detecting-circles-images-using-opencv-hough-circles/.

[8] Ultralytics. *Getting Started*. URL: https://docs.ultralytics.com/quick-start/.

[9] A-L. Nozieres. *YOLOv5, End-to-End object detector project on custom dataset*. 2021. URL: https://towardsdatascience.com/yolov5-end-to-end-object-detector-project-on-custom-dataset-5d9cc2c95921.

[10] A. Rosebrock. *Measuring distance between objects in an image with OpenCV*. 2016. URL: https://www.pyimagesearch.com/2016/04/04/measuring-distance-between-objects-in-an-image-with-opencv/.

[11] URL: https://project.dke.maastrichtuniversity.nl/SwarmLab/.

[12] R. Harvey. *Lecture 10: Hough Circle Transform*. 2005. URL: https://www.cis.rit.edu/class/simg782/lectures/lecture_10/lec782_05_10.pdf.

[13] H. Zhengzu. *Find peak location in 2-d array*. 2016. URL: https://stackoverflow.com/questions/36652117/find-peak-location-in-2-d-array.

[14] G. Karimi. *Introduction to YOLO Algorithm for Object Detection*. 2021. URL: https://www.section.io/engineering-education/introduction-to-yolo-algorithm-for-object-detection/.

[15] Manishgupta. *YOLO — You Only Look Once*. 2020. URL: https://towardsdatascience.com/yolo-you-only-look-once-3dbdbb608ec4.

[16] G. Jocher et al. *yolov5*. 2022. URL: https://github.com/ultralytics/yolov5.

[17] P. Skalski. *alpha makesense*. URL: https://www.makesense.ai/.

[18] Neural Magic. *Deep Sparse*. URL: https://neuralmagic.com/.

[19] Neural Magic. *CPU inference engine that delivers unprecedented performance for sparse models*. 2022. URL: https://pythonrepo.com/repo/neuralmagic-deepsparse.

[20] A. Rosebrock. *Measuring distance between objects in an image with OpenCV*. 2021. URL: https://www.pyimagesearch.com/2016/04/04/measuring-distance-between-objects-in-an-image-with-opencv/.

[21] Wikipedia. *Precision and recall*. URL: https://en.wikipedia.org/wiki/Precision_and_recall.

[22] P. Huilgol. *Precision vs. Recall – An Intuitive Guide for Every Machine Learning Person*. 2020. URL: https://www.analyticsvidhya.com/blog/2020/09/precision-recall-machine-learning/.

[23] S. Yohanandan. *mAP (mean Average Precision) might confuse you!* 2020. URL: https://towardsdatascience.com/map-mean-average-precision-might-confuse-you-5956f1bfa9e2.

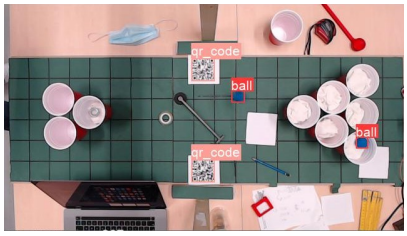
[24] B. Knox. *What is the mAP metric and how is it calculated?* 2018. URL: https://stackoverflow.com/questions/36274638/what-is-the-map-metric-and-how-is-it-calculated.

[25] J. Brownlee. *How to Use ROC Curves and Precision-Recall Curves for Classification in Python*. 2018. URL: https://machinelearningmastery.com/roc-curves-and-precision-recall-curves-for-classification-in-python/.

APPENDIX

Table X gives an example of an image we used for training with the different resolutions.

TABLE X: Different Resolutions Images Example

Resolution	Image
160x90	
320x180	
640x360	

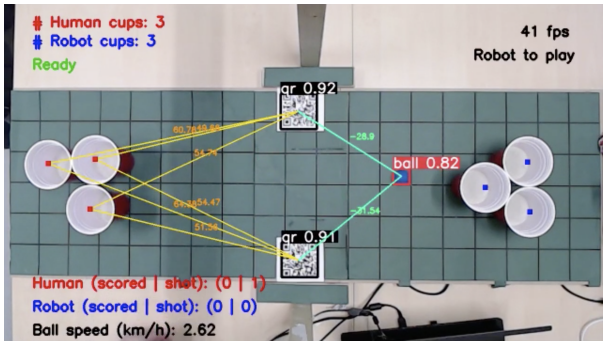


Fig. 25: Full Computer Vision example

TABLE XI: Sparseml mAP over Epochs

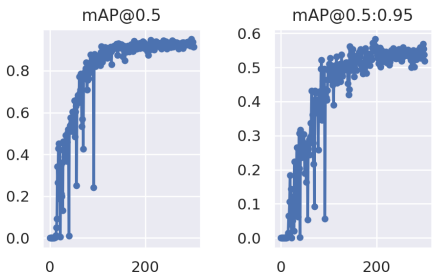
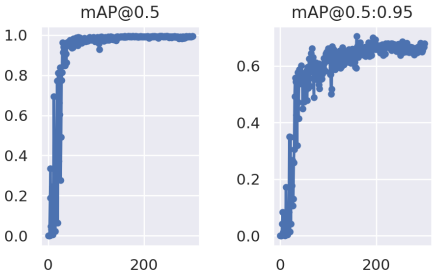
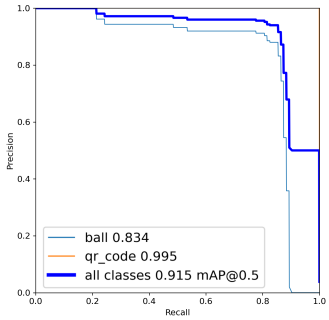
Resolution	Graph
160x90	
640x360	

TABLE XII: Sparseml Precision Recall Curve

Resolution	Graph
160x90	
640x360