# Drawing robot playing Tic-Tac-Toe

**Department of Data Science and Artificial Intelligence**
**Maastricht University**

**2021/2022**

Adele Imparato (i6209306), Jean Janssen (i6211969), Matthijs Kusters (i6214101),
Stijn Overwater (i6200279), Daniel Tossaint (i6221415)

***Abstract*—Because of steady hardware improvements, there are starting to exist certain opportunities in the field of robotics which were previously inconceivable. However, the general public has no way to become informed about the current state and possibilities of the field. Maastricht University has developed several Educational Modular (EDMO) robots, one of these being such a robotic arm with a pen attached to it. The main objective of this research project is to enable this 4 degree of freedom (4dof) robot to safely play the interactive game Tic-Tac-Toe against a human player, using software. In the final product, one has been able to use grid, object and motion detection to gain knowledge about the environment and forward and inverse kinematics to be able to act on this environment with the robot. More importantly, this paper is focusing on the performance of the robotic arm at performing tasks, as well as the robustness of the computer vision used to detect the environment. The results achieved are such that it was made possible to reach great precision when it comes to detecting the grid of the game or the symbols used to play. Moreover, the robotic arm has succeeded to draw lines accurately enough such that it could then draw pluses and squares in order to play the game.**

## I. Introduction

Today, robotic arms are machines much in demand due to their abilities of accomplishing human tasks faster and more precisely. Indeed, we humans like to make our daily life more comfortable by using such technology. This is why robotic arms find their place in many different fields always having the same goal: to improve the efficiency to accomplish the task for which they are used for. This is also the case of the robotic arm able to play interactive games such as Tic-Tac-Toe [10]. The advantage of using a robotic arm to play against a human player is that the robot can play with different levels of difficulty thanks to the game AI, but also, it plays fair and is precise when drawing pluses or squares[1].

Within this project, software for a 4dof robotic arm was developed in order for the robot to be able to play the interactive game Tic-Tac-Toe against a human player. The overall project was code in Python and the software Arduino [1] was used in order to control the robot. Components of the robot discussed in this paper include: Computer vision for analyzing the game through the use of a camera, as well as collision detection if the trajectory of the robot or the playing field is disturbed. Kinematics, both forward and inverse, for checking if there is any disturbance in the robot's trajectory as a collision prevention measure and executing that trajectory. And lastly, game AI in order to play the game with the other player.

[1]NB: For convenience, the robot draws pluses and squares instead of crosses and circles although the human player sketches crosses or circles.

### A. *Problem description*

As mentioned in the introduction, the goal of this project is to build software for a 4dof robotic arm able to play Tic-Tac-Toe against a human player. The challenges of this project are to get computer vision that is robust enough to detect a grid, crosses and circles sketched on the board game, in a way that the accuracy reach is high enough such that a game can be played without any detection issues. Another challenge is the precision of the arm in terms of reaching the correct position. Indeed, it may seem simple to ensure that the robotic arm reaches the end-effector position and sketches, but in fact, gravity, motor speed, change in motor angle and the force applied by the pen on the board all play an important role. Therefore, this paper will aim to answer the following research questions:

1) **RQ1: How accurate is the deep neural net described at categorizing cells compared to the base model CNN (see section IV.B.3)?** For detecti In order to improve the neural network, one adds at what lev convolutional layers to the neural net until it reaches shapes, the high performance.
2) **RQ2: For the computer vision, what is the optimal method for thresholding with regards to adaptive method and blocksize?** The application needs to binarise the board robustly enough such that noise interference originating from inconstant lighting will not interrupt the game in progress. To test how robust our grid detection is, we will change the adaptive method as well as the blocksize and see how well the computer vision detects the grid.
3) **RQ3: How robust is the robotic arm at drawing lines?** The robotic arm needs to be able to

sketch pluses and squares precisely enough so that the human player understands that the robot made a move.

To do so, this paper will first describe the hardware and setup used for the purpose of this project. Then, it will introduce similar researches that have been done over the same topic. After that, three research questions will be articulated and the methods used to develop the software will be described. These methods are divided into four categories: the kinematics (both forward and inverse), the computer vision, the game AI and eventually the state machine. Eventually, this paper will describe the development and results of three different experiments that allow to answer the three research questions previously mentioned.

## II. Hardware and setup

Figure 1 is a picture of the hardware which mainly consists of a setup made of wood and the robotic arm. As indicated on the picture, the setup is made of:

1) a wood plank on which the whole setup is attached to;
2) a whiteboard where the game will be sketched on;
3) a pulley to which a nut is attached to, enabling to counter the gravitational force;
4) a camera (Live! Cam Sync HD, 720p) able to visualize the current state of the game;
5) an umbrella that prevents any issue related to light reflection on the whiteboard, and finally;
6) the robotic arm which consists of 4 EDMO modules [3] attached together as well as a pen that allows one to write.

The robot has four degrees of freedom and can be treated as if the robot moves on a plane using 3 planar pitch DOFs rotated in a 3D space using the bottom EDMO motor. The first pitch has a range between -20 degrees and 70 degrees, the second one has a range between 10 and 100 degrees, and the last pitch rotates between -90 and 90 degree angles. Using this setup, the robot can draw on a whiteboard that is parallel to the floor. The is 30.5x42.5 cm2 and is longer on the axis parallel to the wooden the pulley is attached to. The maximum length of the robotic arm is 37 centimeters. This means the robot cannot reach the top part of the board, but as later shown in experiments can reach the majority of the board and is capable of filling in a grid for playing Tic-Tac-Toe.


Figure 1: The complete setup of the robotic arm.

## III. Related work

Naturally, this research paper is not the first to discuss the functioning of a robotic arm playing Tic-Tac-Toe. Indeed, several similar works have already been conducted. Here we highligth the following:

| Projects and Publications | Connection to and differences with regard to our project |
|---|---|
| A NAO robot playing tic-tac-toe, Comparing alternative methods for Inverse Kinematics, Renzo Poddighe and Nico Roos, DKE [6] | This research paper focuses on the comparison of two distinct methods for the Inverse Kinematics used to enable a NAO robot to play Tic-Tac-Toe. It highlights the fact that the FABRIK algorithm works best in order to control their robot, which one is different from ours since it is humanoid. In our research, we develop the kinematics methods used in the case of a robotic arm made of EDMO modules. |
| Drawing robotic arm, Maria Markovska and Felica Gihl Vieider [4] | This thesis describes how to design a robotic arm and investigates which drawing techniques are best in order to make it draw. Also, it analyses which controller and PID is the most suitable for their robot. Thus, this paper mainly focuses on the drawing techniques of the robotic arm while our project also focuses on the methods used to make it play an interactive game, namely Tic-Tac-Toe. |
| Substantive Tic-Tac-Toe : A realistic way to play tic-tac-toe games with Intelligent Robotic Arm, Md. Mehedi Hassan Rupok [2] | This research paper focuses on the game Tic-Tac-Toe and how to make the robotic arm play correctly. This paper is really close to ours since it highlights all the methods used to make the robot play Tic-Tac-Toe. However, the main difference is that their robot is a 3dof arm while ours is a 4dof arm which has a great influence over the kinematics. |

## IV. Methods

### A. *Kinematics*

The kinematics of the robot allows for it to move to the desired positioning of the pen. Figure 2 illustrates the robotic arm in a schematic way.

*1) Inverse Kinematics (IK):* In order for the robot to reach a desired end-effector location $X_{EE} = (x, y)$, an approach using analytical kinematics was implemented. A total of four angles for each of the DOFs is computed and then combined into a command that is sent to the robot. These four angles will be referred to as $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$ respectively where $\theta_4$ until $\theta_1$ are the angles from the end-effector to the base of the robot. Firstly, $\theta_1$ is calculated since this joint is the only one having any impact on the x-coordinate of the end-effector. It is calculated to orient the robot to the desired position using the following equation.

$$\theta_1 = atan^2(x, y) \tag{1}$$

Then, one calculates the length of the radius from the zero point, the base of the robot, to the end-effector location and assign this length to $py$. The zero point can be defined as the center of first EDMO module responsible for the yaw. Relative to this point, the board is located at (0, 10, 1). This point is the middle of the bottom edge of the board. This $py$ is the length that the robot needs to reach with the other joints to come to the desired location on the $x, y$ plane (See figure 3). To simplify the inverse kinematics, one assigns the value of $\theta_3$ according to this condition: If $py$ is larger than or equal to 25, meaning the end-effector location is in the top half of the board, $\theta_3$ is set to 50 degrees. If $py$ is smaller than 25, $\theta_3$ is set to 95 degrees.

The fact that $\theta_3$ is set to one of these two numbers, means that the length of the line from the joint on $\theta_2$ and the joint on $\theta_4$ is constant and one can compute it. We call this length $l_a$. Thus that means that by calculating the radius from $\theta_2$ to the end-effector, one can create a triangle with this length $l_c$, $l_a$ and $l_4$, the length from $\theta_4$ to the end-effector position. We can also create a triangle with sides $l_a$, $l_2$ and $l_3$, $l_2$ and $l_3$ being the length from $\theta_2$ to $\theta_3$ and $\theta_3$ to $\theta_4$, respectively. Because of the fact that we know the length of all sides of both triangles, one can use The Law of Cosines [7] to compute any angle of these triangles. Thus, one can compute $\theta_2$ and $\theta_4$ using the following formulas:

$$\theta_e = acos((l_2^2 + l_a^2 - l_3^2)/(2l_2l_a)) \tag{2}$$

$$\theta_b = acos((l_a^2 + l_c^2 - l_4^2)/(2l_al_c)) \tag{3}$$

$$\theta_f = atan(z/py) \tag{4}$$

$$\theta_2 = 90 - (\theta_e + \theta_b + \theta_f) \tag{5}$$

$$\theta_c = acos((l_a^2 + l_b^2 - l_c^2)/(2l_al_b)) \tag{6}$$

$$\theta_d = acos((l_a^2 + l_3^2 - l_2^2)/(2l_al_3)) \tag{7}$$

$$\theta_4 = 180 - (\theta_c + \theta_d) \tag{8}$$

For the visualization of these calculations, see figure 2.

*2) Forward Kinematics (FK):* Another type of kinematics, the forward kinematics, is used when the values of the four joint angles – $\theta_1$, $\theta_2$, $\theta_3$, $\theta_4$ – are sent to the motors and one aims to find the end-effector position. Forward kinematics works the following way: one takes the formulas for calculating the end-effector position (in the appendix, see figure19) and then continue to extend those formulas such that it would fit to the system of the robotic arm which has 4 DOF. Since the starting position of the arm is up straight, the plane from which we calculate the end-effector plane is vertical. That is why the height $z$ is to be calculated using cosine, and the length of the arm into the board with sine, after which one can split those in

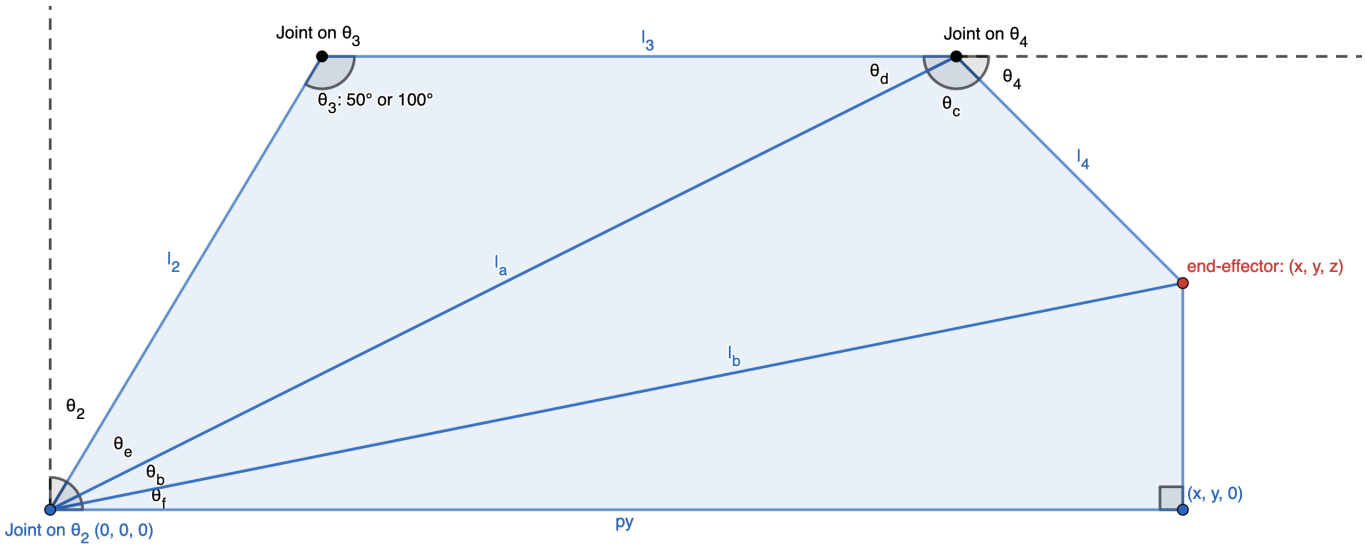


Figure 2: Calculation of $theta_2$, $theta_3$ and $theta_4$.

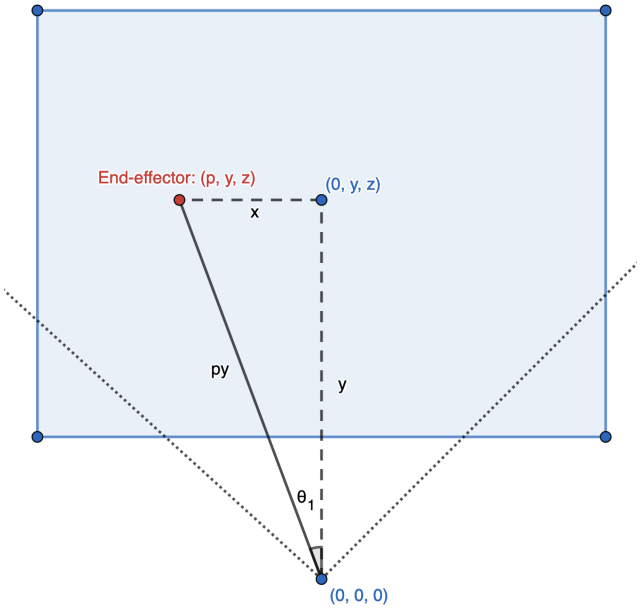


Figure 3: Schematic of $theta_1$ and py

an $x$ and $y$ coordinate using sine and cosine respectively. Eventually, the formulas for calculating the end-effector position look as follows:

$$x = (l_2 \cdot sin(\theta_2) + l_3 \cdot sin(\theta_2 + \theta_3) + l_4 \cdot sin(\theta_2 + \theta_3 + \theta_4) + l_{pen} \cdot sin(\theta_2 + \theta_3 + \theta_4 + 90°)) \cdot sin(\theta_1) \quad (9)$$

$$y = (l_2 \cdot sin(\theta_2) + l_3 \cdot sin(\theta_2 + \theta_3) + l_4 \cdot sin(\theta_2 + \theta_3 + \theta_4) + l_{pen} \cdot sin(\theta_2 + \theta_3 + \theta_4 + 90°)) \cdot cos(\theta_1) \quad (10)$$

$$z = l_2 \cdot cos(\theta_2) + l_3 \cdot cos(\theta_2 + \theta_3) + l_4 \cdot cos(\theta_2 + \theta_3 + \theta_4) + l_{pen} \cdot cos(\theta_2 + \theta_3 + \theta_4 + 90°) \quad (11)$$

### B. *Computer vision*

Another important part of this project is the computer vision. It concerns all the methods needed in order to detect the game state using the camera.

*1) Image processing:* Before analyzing the game state, the image is enhanced to make the system more robust. First, the game board is isolated from the rest of the picture by cropping out the sides and converting the cropped image to gray scale.

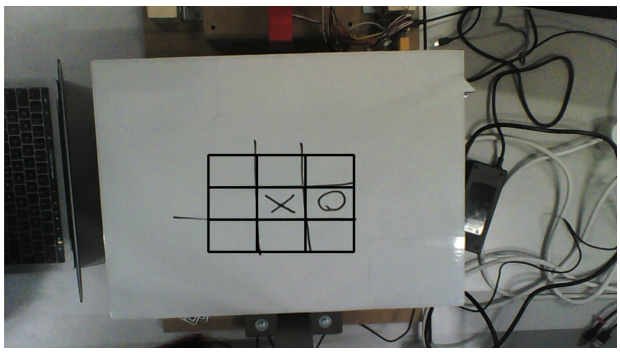

Figure 4: Starting webcam image.

Figure 5: Visual representation of the problem with simple thresholds.

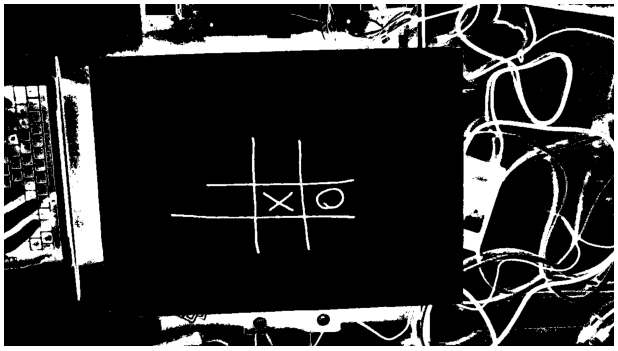

Figure 6: Visual representation of the problem with adaptive Gaussian threshold.

To further increase robustness, the image undergoes a threshold process with the use of an adaptive threshold. Using an adaptive threshold instead of a simple universal threshold allows for different calculated threshold values for several smaller regions as opposed to one for the entire picture. This results in a clear binary representation of the game board.

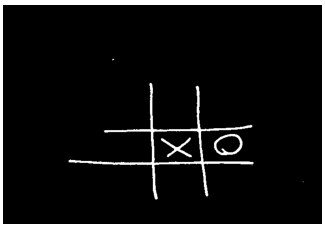

Figure 7: Adaptive Gaussian threshold isolates game board

*2) Analysing game state:* Now that a clear representation of what is going on has been established, the game state of the board can be analyzed, and the next move determined. The game state is represented by an array consisting of nine entries. The integer 0 represents an empty slot, 1 represents a slot that is occupied by an X symbol and 2 represents a slot that is occupied by an O symbol. Now, utilizing the so-called 'Find Contours' function from the library OpenCV [5], the grid is extracted from the image. This is done by sorting all contours in the image from large to small. With the assumption that the grid is the second biggest contour in the image, the bounding box of this contour is returned, allowing us to find the location of the grid. Simple operations are performed in order to identify the size of one of the nine cells of the grid in the image. Once this has been done, a generated 3x3 grid is overlayed onto the image, splitting the box into 9 segments each representing a slot in the original 3x3 grid.

Figure 8: A single cell extracted from the grid.

Then, with the use of the convolutional neural net (CNN), a prediction of the symbol occupying each cell is made. By cross-checking the coordinates of each found cell with the corresponding slot, the array is iteratively filled with each integer and thus, an accurate representation of the board is created. Now that the array represents the game board at hand, a minimax algorithm is used to generate the next move. Another feature of the game allows the adjustment of the implemented slider with a gage from 0 to 100, changing the difficulty of the AI. The value of the slider represents the percentage of minimax used, instead of a random agent.

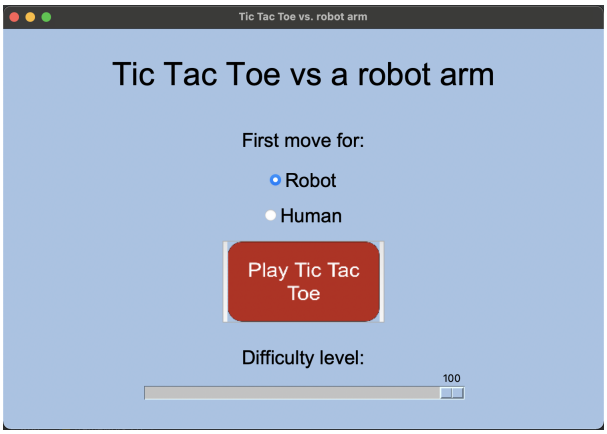


Figure 9: User Interface: starting window with slider defining the difficulty level

*3) CNN:* The base model can be divided into two main aspects: The first being the feature extraction front-end of the CNN which is made up of convolutional layers and pooling layers and the second being the classifier back-end that makes predictions of the input.

The convolutional front-end begins with a single convolutional layer with a relatively small filter of size (3, 3) and a limited number of filters (32). This layer is followed by a max pooling layer after which the filter maps are flattened to provide features to the classifier back-end.

Since the task is set up as a multi-class classification problem, an output layer consisting of three nodes will be required so that predictions on the probability distribution of an image belonging to each of the three classes can be made. A soft-max activation function is used after the flattening, integrated between the feature extractor and the output layer. A dense layer is also added in order to interpret the features from the extractor layer.

For the stochastic gradient descent optimizer a learning rate of 0.02 is used along with a momentum of 0.9. The aim is to optimize the categorical cross-entropy loss function. This method is suitable for multi-classification and the classification accuracy metric implemented later in the paper since the same number of examples is used for each of the three classes.

All layers, except for the final soft-max layer, will use the ReLU activation function [8] and the He weight initialization.

With a well-established base model, the feature extraction front-end can be deepened. This is done by following VGG-like pattern. The innovative CNN model is named after the Visual Geometry Group at Oxford [9] and its architecture is comprised of a repetition of the so-called VGG blocks. These blocks are groups of convolutional layers with small filters (3x3) that are followed by a max pooling layer.

With this architecture in mind, the model at hand was adjusted to incorporate an increasing number of filters by adding more convolutional and pooling layers with the same sized filter. Specifically, two more convolutional layers with 64 filters each were added, followed by another max pooling layer.

This concludes the outline of the model as a whole and leads to its evaluation. For the evaluation of the model a five-fold cross-validation will be used. The value of $k = 5$ has been chosen to limit the run time but still provide a sufficient baseline for repeated evaluations. Each test set will be comprised of 20 percent of the training set. Given that the training set is made up of 5000 images, the test set will be made up of 1000 images.

Before the training dataset is split, it is shuffled. Sample shuffling is performed each time to assure that any model that is being evaluated has the same train and test sets in each fold. Thus, providing direct model to comparisons.

### C. Motion detection

For the motion detection, the following phases will be discussed: grayscale conversion and noise removal, subtraction operation between the background and the foreground, threshold application and, finally, detection of contours. These phases will allow us to detect movement using OpenCV.

*1) Grayscale conversion and noise removal:* Before preforming any operations on the frames they are converted to grayscale. This conversion creates images that are less complex which is beneficial for the purpose at hand. Noise in the image that has been created by the camera and the lighting is also minimized. This process consists of averaging each pixel with its neighboring pixels, otherwise known as smoothing.

*2) Background subtraction:* Background subtraction is the process of taking a frame of the video stack without movement, usually the first frame, and calculating the difference between this frame and successive frames that are picked up by the webcam. The frame without movement is regarded as the background. The frame of interest is subtracted from the background and the result is an image with a black background where motion can be detected.

Figure 10: Absolute difference between background and foreground

This simple technique does not require the object to have something that identifies it like a QR code. However, a disadvantage is that if an object has a similar color to the background, movement is poorly detected. Additionally, background subtraction is highly sensitive to changes in lighting and shadows. A solution to this is that instead of taking the first frame in the stack as background, the weighted mean of the previous frames along with the current frame is taken. Then, the weighted average is subtracted from the current frame, resulting in what is coined the frame delta. This entails that the background can be adjusted dynamically as the lighting conditions change. Now, that the delta has been obtained the resulting image can undergo a thresholding process to obtain a significant difference from the background model.

*3) Threshold application:* Upon producing the difference frame, the image can be binarised. Thresholding an image is the process of converting all pixel values of below a certain threshold to 0 and all pixels above to 255. Resulting in a binary image. An adaptive thresholding function is used, as mentioned in the previous section on image processing. This function calculates different threshold for each region in the image by looking at the surrounding pixels. This allows for a clear binary image that does not lose any crucial information.

Figure 11: Adaptive threshold of the difference image

*4) Blob detection:* Once the binary image has been created, white spots in the image, called blobs, are detected. The blobs are a set of pixels that have the same value as its neighbouring pixels. This is done by applying contour detection. Contours can be explained simply as a curve joining all the continuous points, along the boundary, that have the same color or intensity [5]. When a blob that is bigger than, for example, 30 pixels is detected it is translated into detected "motion" in the frame and is passed as a message along to the framework. This will in turn shut off the robot arm, preventing it from colliding with the object in question.

### D. *Game AI*

For the Game AI, we wanted to implement a difficulty system. We did this by giving the user a slider in the start screen. By setting this slider to a certain value from 0 to 100, the user sets the game up so that a certain percentage of the time, the computer will use a good AI algorithm to determine its move. All the other times, it will use a random move. The AI algorithm that is used is the well-known minimax algorithm, which consists of building the game tree containing all the states of the game (in total 255168 nodes) and then retrieve the best move out of it, i.e. the move that has the highest chance of making the robot win the game. 1

### E. *State Machine*

Like many other games, Tic-Tac-Toe is a turn-based game. This means, for the specific application of this paper, that one can model the game by using states. Moreover, because of the difference in information needed in different situations in the game, using a state machine is beneficial for using the right aspects of our application and providing the right information at the right time.

When pressing the Play Game button, the game is setup and a loop starts. This loop functions as the game. It starts the video streaming and starts the state machine. Every iteration of this loop, grid detection is run to check for a grid on the board. If one is found, it overlays this grid on the UI and shows that. Also, motion detection is run to check whether the field is empty. If the field is empty, the state machine can run. When running, the state machine consists of several main states.

*1) `begin`:* As the name entails, this state instantiates the game. It checks who plays the first turn of the game, according to the radio buttons from the starting screen. According to this it changes the state to `make_move` (if the computer has the first move) or `wait_move` (if the user has the first move).

*2) `end`:* This state ends the game. By setting the state to this, the condition of the game loop returns false, thus the game loop is discontinued.

*3) `make_move`:* When the turn switches from the user to the computer, this state is used. The computer decides what its next move will be using the Game AI. It converts the coordinates of the box, in which the computer wants to make its move, from computer vision coordinates to coordinates the kinematics can use. It then calls the kinematics to provide a list of commands to provide the Arduino. Once this has been done, the state is set to `moving` to execute the moves.

*4) `moving`:* This state exists to execute the commands that are computed in the `make_move` state. It checks whether a certain amount of time has passed. If it has, it executes the next move in the output list according to an index and then increments this index. It does this by sending a string to Arduino, which then controls the motors. Afterwards, it sets the delay as the amount of time that has to pass for the next move to be made. All this could normally be done in a quick loop, iterating over the list. However, this would mean that the entire move is executed in one iteration of the game loop. By having this state, one can easily check the motion detection, since that runs every iteration of the loop. Once the state machine has iterated over the loop, thus executing the move, it checks if the game has finished. If it has, the state is set to `end`. If it has not, the state is set to `wait_move`, to let the user play their move.

*5) `wait_move`:* When it is the human player's turn to move, the state machine is in this state. It loops, continuously showing the field, until the space bar is pressed. Once the space bar is pressed, the symbol detection will look at all previously empty boxes in the field and, by using symbol detection, see if there has appeared a symbol. Once it has detected a new symbol it updates the board accordingly. If the game has finished, the state is set to `end`, otherwise the state is set to `make_move`, to let the robot make the next move.

*6) Issues arisen:* When all components come together, this is how the state machine should work. However, several parts were not managed to connect properly to the framework of the application. Notably, dealing with the buffer of the Arduino turned out to be a frequent issue. Furthermore, the equations for coordinate conversion from video streaming to our kinematics coordinate system did not function due to constant deviations in the video frame orientation. This could be due to the different operation systems used to develop this project. Because of these problems, the experiments are conducted with separate parts of the program and no experiments could be executed on a system level.

## V. Experiments

For this project, several experiments have been conducted:

1) Experiment 1: Determine how accurate the deep
Adding layers to the convolutional neural network and see how it improves its performance by measuring the accuracy, cross-entropy loss and standard deviation.

   For this experiment, we will train the baseline model for a modest 20 training epochs with a default batch size of 32 examples. The test set for

each fold will be used to evaluate the model both during each epoch of the training run, so that we can later create learning curves, and at the end of the run, so that we can estimate the performance of the model. As such, we will keep track of the resulting history from each run, as well as the classification accuracy of the fold.

2) Experiment 2: Determine the optimal parameters used in the computer vision by changing the blocksize and adaptive method and see how it performs. (in response to RQ2)

   The blocksize is the size of a pixel neighbourhood that is used to calculate a threshold value. This experiment consists of comparing different adaptive methods and blocksizes. The two adaptive thresholding methods used here are: the adaptive threshold mean C, which is the mean of the neighbourhood area minus the constant C, and the adaptive threshold Gaussian C, which is a Gaussian-weighted sum of the neighbourhood values minus the constant C.

3) Experiment 3: Determine how robust the robotic arm is by making it draw several lines and see how it performs. (in response to RQ3)

   In order to test the performance of the robotic arm, it is necessary to make it do the same task several times. For this experiment, the robot sketched 24 parallel horizontal lines and 26 parallel vertical lines, the goal being to make it sketch the lines as best as possible. The lines are 3cm long, they correspond to the lines needed to build a plus or a square. For this reason, we asked the robot to sketch lines relatively in the middle of the whiteboard, where it will play the game.

## VI. Results

### A. Experiment 1

As it can be seen on figure 12, 13 and 14, the accuracies reached are above 99% and the cross entropy losses are below 0.1.

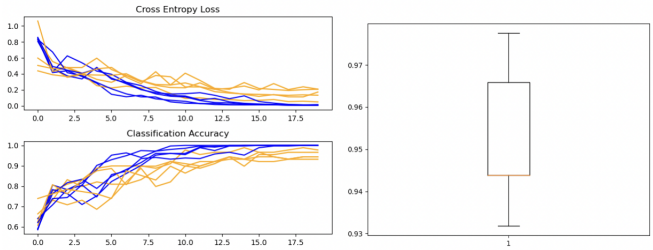


Figure 12: Epoch = 20, basemodel CNN Accuracy: mean=95.258 std=1.664, n-folds=5

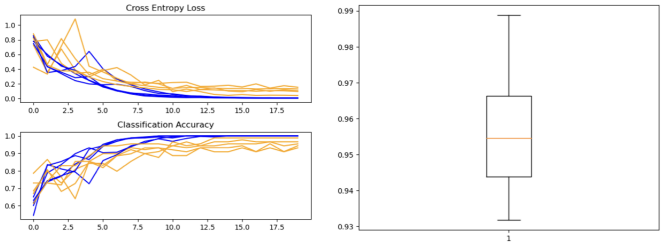


Figure 13: Epoch = 20, deepe CNN Accuracy: mean=95.705 std=1.954, n-folds=5

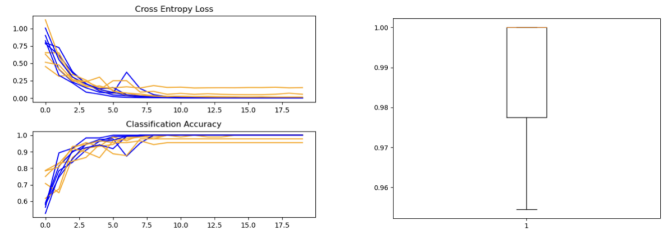


Figure 14: Epoch 20, Deepest CNN Accuracy: mean=98.641 std=1.816, n-folds=5

### B. Experiment 2

In figure 15 and 16, $C = 2$ for all plots so that it makes it easy to see the effect of changing the blocksize.

In this second experiment, one can see that the pictures part of figure 16 (Gaussian threshold) have less noise than the one from figure 15 (mean threshold).

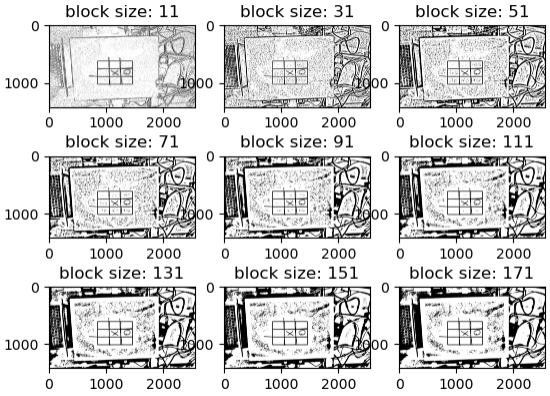


Figure 15: Mean threshold according to blocksize

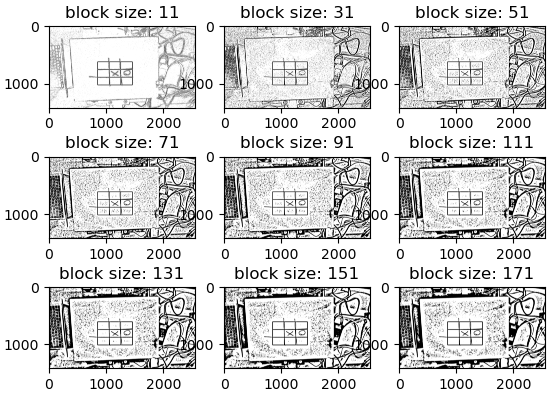


Figure 16: Gaussian threshold according to blocksize

### C. Experiment 3

So that we can get a slightly clearer view of things, a grid has been added in figure 17 and 22 allowing to see how straight the lines were sketched. In the appendix, figure 24 and 23 are representing the lines expected at the correct coordinates. By comparing the expected lines with one obtained, it is possible to evaluate the performance of the robotic arm. Horizontal lines were drawn at a coordinates $y = 10, 15, 20, ..., 35$ and coordinates $x1, x2$ where for each line a space of 5 centimeters was reserved with 1 centimeter spare on each side. So, for $x1 = 1$ and $x2 = 5$, a line is drawn from $x1 = 2tox2 = 4$. These lines are drawn between a minimum of $x = -15$ and a maximum of $x = 15$. For the vertical experiments we flipped the scenario. Lines were drawn at coordinates $x = -15, -10, -5, ..., 15$ and 5 centimeters of space was reserved for a minimum of $y = 15$ and a maximum of $y = 35$.

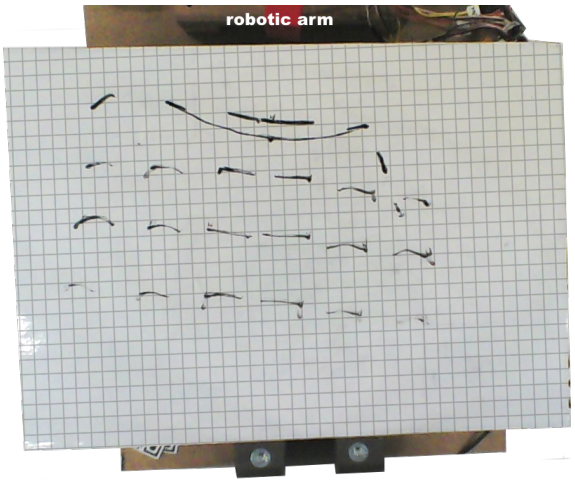


Figure 17: 24 horizontal lines sketched by the robotic arm.

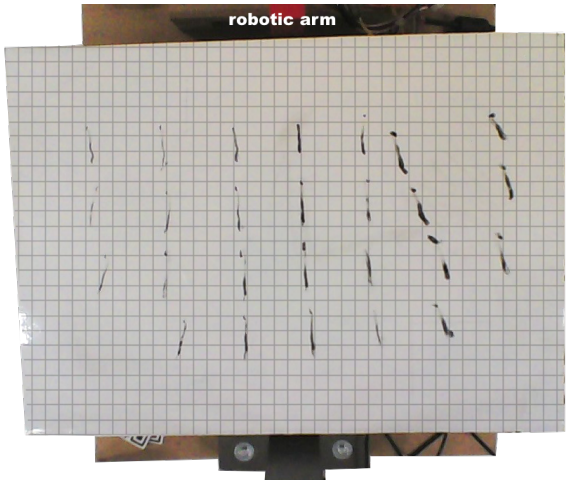


Figure 18: 26 vertical lines sketched by the robotic arm.

## VII. Discussion

### A. Experiment 1

Looking at the base model, one can immediately see through the accuracy metric that there is some clear overfitting and the learning rate is a bit high. The graphs may even suggest that training epochs further could be helpful. In the next model one tries to combat this and as it can be seen, the learning rate has improved and so the amount of overfitting has decreased. There is also a slight increase in accuracy and our standard deviation has also increased. In the last model the highest accuracy is achieved, and the lowest standard deviation. The model has a great learning curve and shows no clear signs of overfitting. This means the deep neural network is the most accurate. In fact, it is robust enough to detect a wide range of symbols X's and O's, which means the symbols sketched by the human player can be distorted and have different sizes, they will still be detected by the computer vision.

### B. Experiment 2

Experiment 2 shows that, in depend of the adaptive method, an increase in blocksize while increase the amount of noise per block. However, it also helps getting a clearer representation of bigger features in the frame which results in a more robust finding of the grid but with the down side of having a lot of noise, which will get in the way of detecting symbols (smaller features) in the frame. Therefore, for functions like detecting symbols, a small blocksize is beneficial as it partitions the frame in to smaller size blocks and thus calculates a new threshold for a smaller partition this increases robustness resulting in a reduced amount of false positives. Furthermore, with regards to the adaptive method, it can be observed that the Gaussian thresholding gives a more coherent frame rather than the mean thresholding. This is probably due to the fact that a Gaussian weighted sum utilises a Gaussian window technique which has a smoothing effect that filter out some of the noise.

### C. Experiment 3

Experiment 3 shows that the robotic arm is able to sketch lines as encoded. However, on figure 17, the six lines that are the closest to the robot are not accurate enough: the lines are not horizontal and the middle lines come together as one. Moreover, some lines are not completely horizontal but rather curvy and the bottom right and left lines almost do not appear. Globally, figure 17 shows that the robotic sketches the lines accurately when centered on the board. For the vertical lines, the same phenomenon can be observed: the robotic arm sketches the lines accurately when relatively centered on the board. Here as well, some lines are not completely straight but rather curvy. Due to the maximum length of the robot being 37 centimeters a line from $(y1, y2) = (34, 31)$ could not be drawn because it was out of reach on both sides of the x-axis.

Therefore, we can reply to RQ3 by saying that the robotic arm can draw lines accurately enough such that a complete game of Tic-Tac-Toe could be played although its robustness could be improved.

## VIII. Conclusion

To conclude this work, researching about the performance of a robotic arm play Tic-Tac-Toe enabled to evaluate the difficulty of getting high precision and accuracy. Eventually, it was made possible to sketch a plus and a square using the robotic arm with a precision that is sufficient in order for the human player to see the computer's move. Furthermore, it had been possible to get the computer vision strongly robust such that the grid and symbols made by the human player can be detected by the camera in almost 100% of the situations. For further studies, a solution for the state machine would be researched as the reason whether this does not function is not apparent. Moreover, the drawing of the robotic arm could be more precise by adding cases to the third $\theta$ calculated. Furthermore, better calibration and control of gravity could make for better pressure management on the pen in order for drawing by the robot to become less vague. Added to that, allow the robot to draw circles and crosses as an expansion upon the pluses or squares. For the computer vision, further studies could include experimentation with the motion detection as well as implementation of object detection of the board. Where, the objects placed cause the board to be disturbed and the robot to stop in its trajectory. The board should not interfere with this object detection.

## References


[1] Arduino Software. 2022. URL: https://www.arduino.cc/en/software/.

[2] Mehedi Hassan. Substantive Tic-Tac-Toe : A realistic way to pl Nov. 27, 2016. URL: https://www.academia.edu/30113113 / Substantive _ Tic _ Tac _ Toe _ A _ realistic_way_to_play_tic_tac_toe_games_with_Intelligent_Robotic_Arm.

[3] Laboratory for Cognitive Robotics and Complex Self-Organising Systems. EDMO – DKE SwarmLab. 2020. URL: https://project.dke.maastrichtuniversity.nl/SwarmLab/edmo/.

[4] Maria Markovska. Drawing robotic arm. 2017. URL: http://www.diva-portal.org/smash/record.jsf?pid=diva2%3A1200466&dswid=5976.

[5] OpenCV, Open Source Computer Vision. 2022. URL: https://docs.opencv.org/3.4/index.html.

[6] Nico Roos. A NAO robot playing tic-tac-toe Comparing alterna Jan. 1, 2013. URL: https://www.academia.edu/34513317/A_NAO_robot_playing_tic_tac_toe_Comparing_alternative_methods_for_Inverse_Kinematics.

[7] The Law of Cosines. 2017. URL: https://www.mathsisfun.com/algebra/trig-cosine-law.html.

[8] Jayant Verma. ReLu Function in Python. Nov. 4, 2020. URL: https://www.journaldev.com/45330/relu-function-in-python.

[9] Visual Geometry Group - University of Oxford. 2022. URL: https://www.robots.ox.ac.uk/%7Evgg/.

[10] wikiHow. How to Play Tic Tac Toe. Nov. 12, 2021. URL: https://www.wikihow.com/Play-Tic-Tac-Toe.

. Appendix

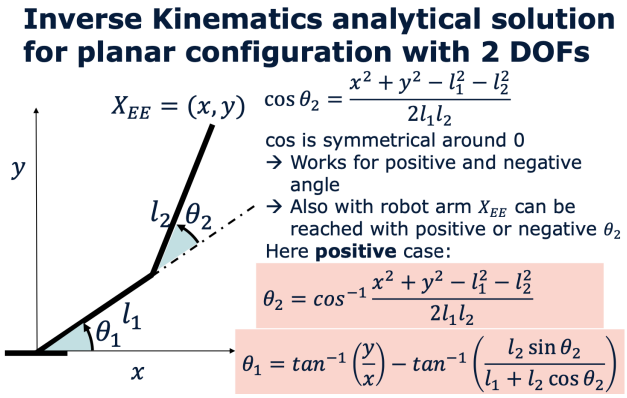


Figure 19: Maastricht University, Department of Data Science and Knowledge Engineering, Robotics and Embedded Systems, Dr. Rico Mockel, Inverse Kinematics analytical solution for planar configuration with 2 DOFs.

**Algorithm 1** Minimax Algorithm

**Require:** $node, depth, maximizingPlayer$
**if** $depth = 0$ **or** node is a terminal node **then**
  **return** the heuristic value of node
**end if**
**if** $maximizingPlayer$ **then**
  $value := -\infty$
  **for** each child of node **do**
    $value$ := max($value$, minimax(child, depth - 1, false))
  **end for**
**else**
  $value := +\infty$
  **for** each child of node **do**
    $value$ := min($value$, minimax(child, depth – 1, true))
  **end for**
**end if**
**return** $value$

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 13, 13, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 11, 11, 64)        18496
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 9, 9, 64)          36928
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 4, 4, 64)          0
_________________________________________________________________
flatten_5 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               102500
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 303
=================================================================
Total params: 158,547
Trainable params: 158,547
Non-trainable params: 0
_________________________________________________________________
```

Figure 20: model summary deep cnn.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 32, 32, 64)        640
_________________________________________________________________
activation (Activation)      (None, 32, 32, 64)        0
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 64)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        18464
_________________________________________________________________
activation_1 (Activation)    (None, 16, 16, 32)        0
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0
_________________________________________________________________
flatten (Flatten)            (None, 2048)              0
_________________________________________________________________
dense (Dense)                (None, 64)                131136
_________________________________________________________________
activation_2 (Activation)    (None, 64)                0
_________________________________________________________________
dropout (Dropout)            (None, 64)                0
_________________________________________________________________
dense_1 (Dense)              (None, 3)                 195
=================================================================
Total params: 150,435
Trainable params: 150,435
Non-trainable params: 0
_________________________________________________________________
```

Figure 21: model summary deepest cnn.

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_5 (Conv2D)            (None, 26, 26, 32)        320
_________________________________________________________________
max_pooling2d_5 (MaxPooling2 (None, 13, 13, 32)        0
_________________________________________________________________
flatten_5 (Flatten)          (None, 5408)              0
_________________________________________________________________
dense_10 (Dense)             (None, 100)               540900
_________________________________________________________________
dense_11 (Dense)             (None, 3)                 303
=================================================================
Total params: 541,523
Trainable params: 541,523
Non-trainable params: 0
_________________________________________________________________
```

Figure 22: model summary basemodel cnn.

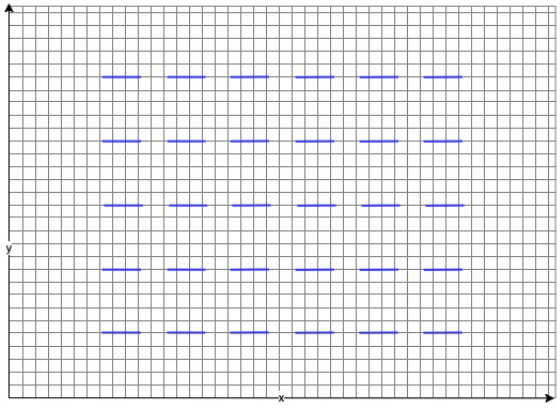


Figure 23: The grid represents the whiteboard, each cell corresponds to 1cm. The blue lines correspond to the 24 expected horizontal lines in experiment 3.

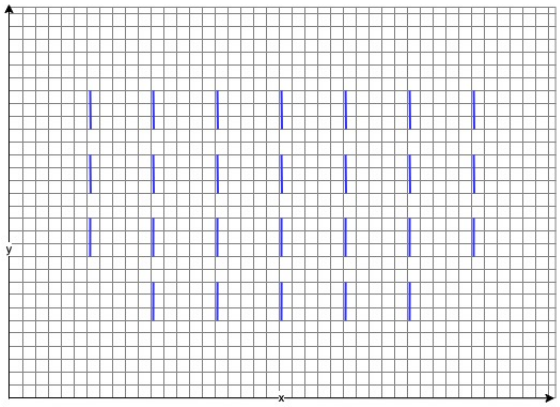


Figure 24: The grid represents the whiteboard, each cell corresponds to 1cm. The blue lines correspond to the 26 vertical lines expected in experiment 3.