Learning Hand-Eye Coordination on NAO and its Applications

Ana Gaja Boc
Jožef Stefan Institute, Department of Knowledge Technologies
Jamova 39, 1000 Ljubljana
Fakulteta za računalništvo in informatiko
Večna pot 113, 1000 Ljubljana
ab9870@student.uni-lj.si

Sara Bertoncelj Čadež
Jožef Stefan Institute, Department of Knowledge Technologies
Jamova 39, 1000 Ljubljana
Fakulteta za računalništvo in informatiko
Večna pot 113, 1000 Ljubljana
sb4914@student.uni-lj.si

ABSTRACT
This paper focuses on learning hand-eye coordination on robot NAO. It elaborates on two different approaches for computing inverse kinematics using neural networks. It also presents two applications, based on the computed inverse kinematics: a system that enables the robot to play tic-tac-toe against a human opponent and a system that enables the robot to replicate simple shapes that it sees.

Keywords
robotics, inverse kinematics, vision recognition

1. INTRODUCTION
Inverse kinematics is commonly used for solving problems such as object grasping, visually guided tasks and also in 3D animation for interaction between characters and other objects in the animated world. While calculating the forward kinematics, that is the position of the end effector based on joint configuration, is a fairly easy problem to solve, inverse kinematics proves to be more challenging because of its multiple solutions.

Traditional methods are computationally expensive, because they rely on constructing and operating on large and complex matrices. Such is the iterative method, which requires the inversion of the Jacobian matrix. There are also alternative solutions that do not require matrices or rotational angles, such as FABRIK [1] (Forward and Backward Reaching Inverse Kinematics). This heuristic algorithm performs simple, iterative operations that gradually lead to an approximation of the solution, by finding the joint coordinates as being points on a line. Inverse kinematics for the NAO robot implemented with the FABRIK algorithm were described by Renzo Poddighe [5], in an article which focuses on a system that enables the robot to play tic-tac-toe, very much like one of the applications presented in this paper. We propose a third approach by calculating the inverse kinematic with neural networks.

2. NAO ROBOT
The first public version of robot NAO was presented in March 2008. Since then six versions of this humanoid have been produced, each having better cameras, CPU, speech synthesis in more languages and better face recognition. For work described in this paper we used NAO version 4.

It has 25 degrees of freedom. The motion ranges of two joints are important for the computation of inverse kinematics: the right shoulder roll which has the motion range from -76 to 18 degrees and the right elbow roll which has the motion range from 0 to 88.5 degrees. It has 1.6 GHz CPU ATOM Z530, 1 GB of RAM and 2 GB of Flash memory. The camera has up to 1280x960 resolution with 60.9 degrees horizontal field of view.

NAO's operating system is based on Linux Gentoo and named NAOqi OS. It has built-in libraries that are needed for the NAOqi Framework, the main software that allows communication between the different modules, programming and information sharing.

3. INVERSE KINEMATICS
Inverse kinematics was calculated with two different approaches for two different implementations. For the game of tic-tac-toe joint positions were calculated for pixels on the image of the gaming surface taken with the robot's camera. For drawing simple shapes joint positions were calculated for x and y coordinates of points on the tablet.

3.1 Neural networks
In both cases inverse kinematics was calculated using a regression neural network. For playing the game of tic-tac-toe, the angles in NAO's arm were measured, by tracking a red pen, while the robot moved it across the gaming surface. Recorded data consisted of pixel coordinates of the tip of the red pen in the image taken by the robot's camera and the shoulder and elbow roll angles. For drawing, the shoulder roll, shoulder yaw, elbow roll and elbow yaw were measured, using a graphic tablet and a stylus pen. While holding the pen the robot's hand was moved around on the tablet surface. As the robot was moving the pen, a program recorded the angle of each of the aforementioned joint and the position of the cursor. There were 279 training samples collected, for playing the game of tic-tac-toe and 10000 training samples for drawing simple shapes. With the second method of recording data we could gather a much more extensive sample size. We could not use the same method for playing tic-tac-toe because the model had to use the position in coordinate system of NAO's camera allowing the program

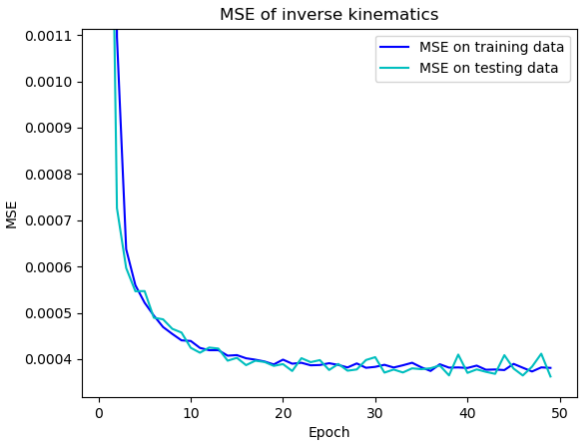

Figure 1: MSE of the neural network for learning inverse kinematics for drawing simple shapes.

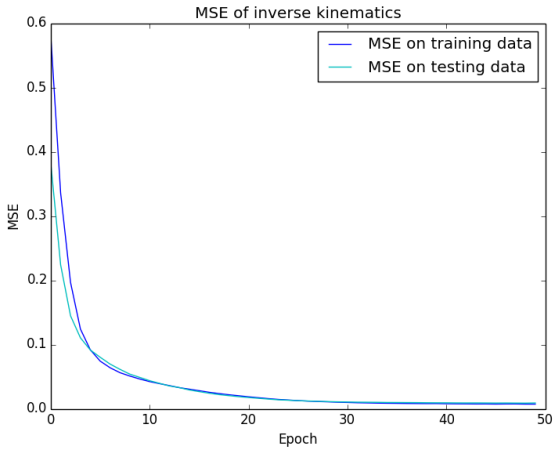

Figure 2: MSE of the neural network for learning inverse kinematics for playing tic-tac-toe.

to work regardless of where in NAO's field of view the playing area was located. Meanwhile the model for drawing was transforming the coordinates from the picture taken with the camera onto a fixed surface in front of the robot. With fewer training samples the model worked better using just two degrees of freedom, meanwhile the model with larger sample size worked better with four degrees of freedom.

Inverse kinematics was calculated using a simple neural network. The neural network was implemented with Keras sequential model. Input variables are x and y coordinates. While neural network for tic-tac-toe had only one hidden layer, neural network for drawing had two identical hidden layers. They had 32 nodes and rectified linear unit activation function. The output layer had two/four nodes which correspond to the dimensions of the output variable (array of two/four angles in radians). To evaluate weights we used the mean square error loss function that calculates the mean error of both/all four angles and the efficient stochastic gradient descent algorithm Adam [3] for optimisation. To train the model we used 50 epochs and a batch size of 10 for the smaller neural network and 50 for the bigger neural network. The mean square error of the final model for drawing was 4.2×10^{-4}, the error at each epoch is shown in Figure 1. Mean square error of tic-tac-toe model was 6.2×10^{-3}, the error at each epoch is shown in Figure 2.

We also calculated inverse kinematics with Support Vector Regression. With training samples for drawing, the mean square error was 1.1×10^{-3}, which is considerably worse than 4.2×10^{-4} error obtained using neural network. With training samples for tic-tac-toe mean square error was 8.1×10^{-3}, while neural network error was 6.2×10^{-3}.

Because drawing requires higher precision, there were more samples collected and a bigger neural network built. It is also because of the large number of samples, that we get higher accuracy by predicting four and not just two angles. For playing tic-tac-toe, precision up to 1 cm is adequate and it can be achieved by predicting just two angles on a smaller data set. Measured precision of inverse kinematics is shown

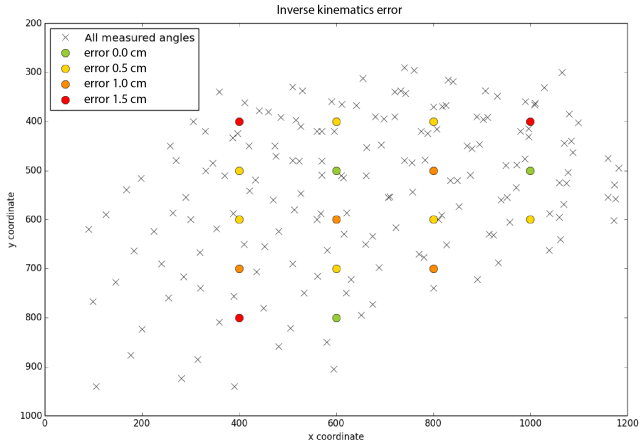

Figure 3: The pixels for which the corresponding arm angles were measured (crosses) and the error of the computed inverse kinematics (dots).

in the Figure 3.

4. APPLICATIONS

We developed two different applications for inverse kinematics. The first one enables NAO to play tic-tac-toe, a simple game where two players take turns in placing their mark (cross or circle) on a grid of size 3 x 3. The player that first succeeds in placing three of his marks horizontally, vertically or diagonally, wins the game. The second one focuses on NAO drawing solid simple shapes, which it captures with its camera.

4.1 Tic-tac-toe

To solve the problem of the robot playing tic-tac-toe, two additional separate modules were developed. A vision recognition module was developed, for recognising the location of the gaming grid and current state of the game. A strategy module was implemented, for deciding which move will most likely lead the robot to victory.

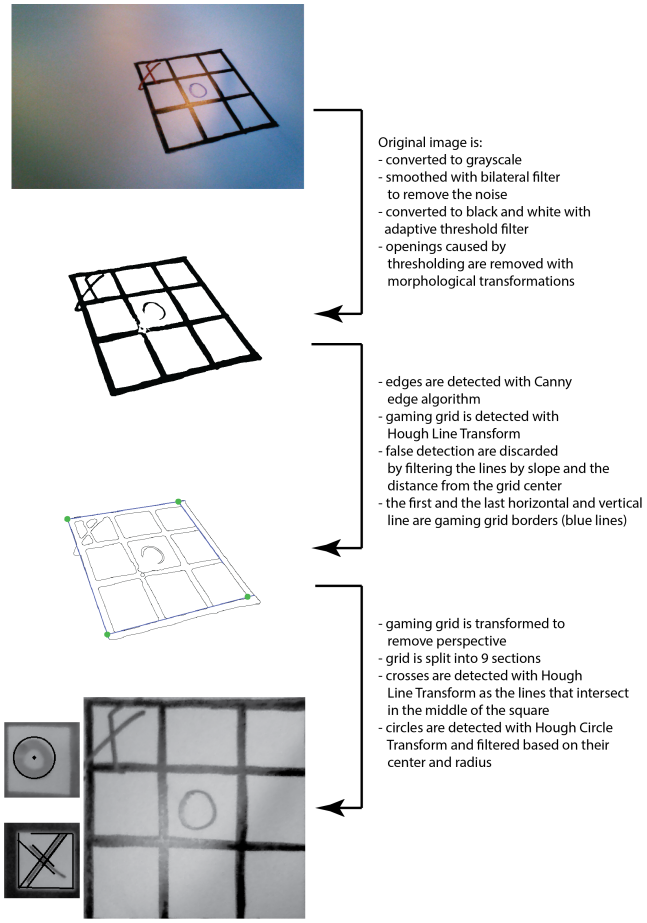

Figure 4: Recognising the state of the game.

The vision recognition component is written in Python using the OpenCV library. Before each move, the robot takes a picture on which the state of the game and the location of gaming board are detected.

The image processing pipeline is shown in Figure 4. Probabilistic Hough Line Transform [4], which returns an array of the start and the end points of all the detected lines is used for the gaming grid detection. These lines are then separated into horizontal and vertical lines and ordered by their position in the image. If there are more than four horizontal or vertical lines detected, they are filtered by their slope and the distances from the previous and the next line. The lines that deviate the most are discarded as false detections. After that, we can be sure that the first and the last horizontal and the first and the last vertical line are the borders of our gaming grid. The intersections of these four lines are also calculated and those four points are used to do a warp transform of the gaming grid, so that the camera perspective is removed and the grid is seen as from a vertical position.

The gaming grid is then split into thirds vertically and horizontally, which gives us nine fields on which there could be a circle or a cross. Hough Circle Transform [7] is used, for circle detection . If a circle is found, its radius and centre are compared to the expected values. Hough Line Transform is used, for cross detection. If there are lines found, possible intersections of these lines are compared to the expected values. If there is an intersection in the middle of the field, then a cross is detected.

In the experiments, the state of the game is correctly recognised in 39/40 cases, which is 97,5% success rate. In 1/40 cases there is an error in vision recognition because of falsely discarding one or more lines as false detections.

The second component of the system is an algorithm that chooses the next move, based on the current game state. Minmax decision rule with alpha-beta pruning was chosen for that, which makes the robot unbeatable at tic-tac-toe.

When the current state of the game is recognised in the image, that information is passed to the algorithm for choosing the next move. Inverse kinematics is then calculated, based on the location of the chosen move. Four arm positions corresponding to the field vertexes are calculated, which are then used for drawing the cross, by connecting the opposite vertexes of the field.

4.2 Drawing simple shapes

In this application our goal is to teach NAO to draw a simple shape that it sees through its camera. Besides calculating the inverse kinematics, another problem we face is computing the points (in the correct order) that the robot must reach to render the shape it was shown. To solve this we use computer vision to process an image that was captured by the robots camera.

When the robot is ready to draw, it will wait to be presented with an image. For the following algorithm to work the image must be of a solid shape on a single-coloured background. When the image is in the field of view of the robot's camera we capture it by pressing on one of the tactile buttons on its head. After the image is captured we need to process it with OpenCV library for Python in order to extract the contours. First the image must be converted from colour image to grayscale. Then a bilateral filter is used to reduce the noise while maintaining defined edges. On the filtered image we can then use canny edge detection [2] to find the edges of the shape we want the robot to draw. From the edge image we can then extract the contours as seen in Figure 5.

When extracting the contours we chose to store all the points along the boundary by not using any chain approximation. This makes the drawing process very slow but it is the most accurate for all types of shapes. Choosing simple or Teh-Chin chain approximation [6] works for curved lines but it can cause problems when drawing shapes with long straight lines. It reduces the number of stored points to mostly just the corner points which means the angles for connecting the points have to be interpolated over a relatively long distance. The angle interpolation of vertical lines produces jagged lines as seen in Figure 6.

After we have extracted the contours we use bounding rectangle to determine the height and width of the shape so we can scale it to fit the robot's drawing area. Once all the

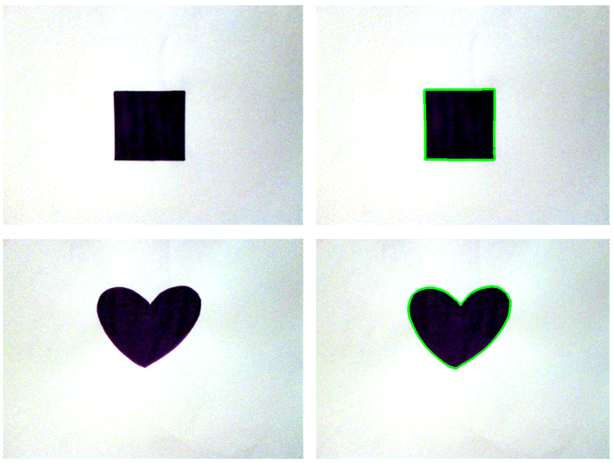

Figure 5: Two pictures NAO captured (on the left) and with extracted contours (on the right).

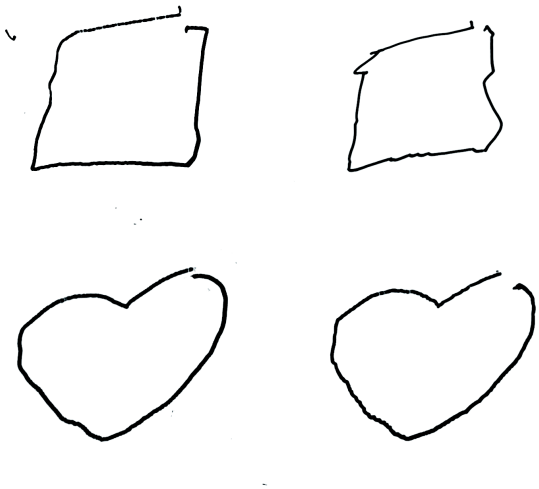

Figure 6: Two drawings produced without chain approximation(on the left) in comparison with two drawings produced with Teh-Chin chain approximation(on the right).

points in the contours are properly scaled we can calculate the angles of robot's joints using the model from the previous section.

Because of the friction between the pen and the drawing surface as well as because of the slight looseness of NAO's joints very small errors accumulate while the robot is drawing. This results in the contours on the render images not being connected at the ends, as shown in Figure 6.

5. CONCLUSIONS

In this paper we solved the problem of hand-eye coordination using neural networks and applied it to two real life problems.

The first system is able to play tic-tac-toe game against a human opponent without losing a single game. Inverse kinematics is precise enough so that the cross that robot draws always has a centre inside the selected field on the gaming grid. Vision recognition correctly recognises the state of the game in 97% of cases. Currently NAO can only draw crosses. We wish to develop the system further so that it will be able to draw circles too. For drawing crosses, four angles that correspond to field vertexes need to be calculated. If the robot were to draw circles, we would need to calculate inverse kinematics for a few dozen positions corresponding to the circle on the field that the robot would draw. The precision of inverse kinematics would also need to be much higher.

The second system successfully extracts contours of solid shapes of one colour and replicates them. By making the image processing more robust we could generalise it to work for more complex drawings. We also wish to improve the processing by combining the current method of contour extraction with other methods of line detection so that the system will be able to draw simple lines that are not joined at the ends.

6. REFERENCES

[1] A. Aristidou and J. Lasenby. Fabrik: A fast, iterative solver for the inverse kinematics problem. *Graphical Models*, 73(5):243–260, 2011.

[2] J. Canny. A computational approach to edge detection. *IEEE Trans. on Pattern Analysis and Machine Intelligence*, 8(6):679–698, 1986.

[3] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *ICLR 2014*, 2014.

[4] J. Matas, C. Galambos, and J. Kittler. Robust detection of lines using the progressive probabilistic hough transform. *CVIU*, 78(1):119–137, 2000.

[5] R. Poddighe. Playing tic-tac-toe with the nao humanoid robot. 2013. url: https://project.dke.maastrichtuniversity.nl/robotlab/wp-content/uploads/Renzo-Poddighe.pdf accessed: 18-August-2019.

[6] C. Teh and R. Chin. On the detection of dominant points on digital curve. *PAMI*, 11(8):859–872, 1989.

[7] H. K. Yuen, J. Princen, J. Illingworth, and J. Kittler. Comparative study of hough transform methods for circle finding. *Image Vision Comput.*, 8(11):71–77, 1990.