Using a robot arm to play drawing-based games

Evelyn Hobert, Emil Korzeń, Eric Risbakk, Hermann Ritter
Department of Data Science and
Knowledge Engineering
Maastricht University
Maastricht, The Netherlands

***Abstract*—A robot arm structure with a pen as its end effector, mounted in front of a table with a writing medium, and two cameras observing the scene in conjunction with two classifiers, a neural network and a c-support vector machine, have been used to recognize doodles, using the Google Quickdraw dataset. The robot arm structure was utilized to play Tic-Tac-Toe. A simulator has been built to study the behaviour of the robot arm structure under the influence of gravity and to model its physical properties. Correction for the gravitational sagging of the robot arm structure has been implemented using data from the simulator. Closed-loop control utilizing noisy sensors has been attempted.**

I. Introduction

Pen robots can be used for a variety of different tasks and activities, be it art, games, or as an assistant. In all of these applications, there are three main challenges:

1) **Recognition**: The robot needs to interact with the writing on the paper, and this writing can be complex: Significant deviations in shape, position, and scale make it harder to identify drawings than just identifying an object which does not change throughout the robot's lifetime.
2) **Production**: The robot needs to draw and write on the paper. Drawing and writing is a task that requires high amounts of precision: The amount of pressure applied by the pen onto the paper has to be controlled, the movements cannot be too abrupt, and the trajectory of the pen has to be stable despite of elasticities present in multiple areas of the robot arm. Any of these factors can produce drawings and writings which are not clean and tidy.
3) **Application**: The drawing and writing has to fulfill a given task.

Our goal was to build a system which combines both art and games, so we thought of two tasks the robot should be able to fulfill:

1) Play Tic-Tac-Toe (Naughts and crosses).
2) Play a doodle recognition and replication game.

The doodle recognition and replication game consists of a human player drawing a doodle onto the writing medium, the robot recognising it, and then replicating it by drawing a similar, but not the same doodle depicting the same object.

To overcome the aforementioned challenges and to fulfill the tasks, the implementation of multiple techniques is required:

1) **Control**: Closed-loop control is needed to achieve precise movements of the end effector.
2) **Computer vision**. Computer vision is needed to observe the writing medium and the end effector, to both be able to provide data for closed-loop control and to support the drawing task through recognition of what has been already drawn and written onto the writing medium.
3) **AI**: Artificial intelligence is needed to process the input data and to produce meaningful output. The complexity of this varies depending on the given task.
4) **Simulation**: The characteristics of the robot arm structure need to be studied to be able to make meaningful predictions and corrections for undesired behaviour such as sagging of the whole structure due to gravity and skipping of the end effector due to friction, as well as wobbling present in the movement of the robot arm.

II. Related Work

Multiple drawing robots have been built in the past. Notable examples are the robots by Megalingam [Megalingam et al.,] and "Paul the drawing robot" [Tresset and Leymarie, 2013]. Both reach a very high drawing accuracy, especially Paul is used to produce art. Both of these robots are planar robots, which is has significant differences in terms of overall system complexity compared to non-planer robots. Non-planar grasping robots like ISAC [Srikaew et al., 1998] have been used to draw, who are more similar to the robot dealt with here. Vieider and Markovska use a robot similar to the one used in here to draw [M. and VIEIDER, 2017]. Robots have also been used to play games, notable examples here are systems like Wizard Chess [Sarker, 2015], which plays Chess using a gripper, and a robot by Barbu et al., which learns to play Tic-Tac-Toe through observation [Barbu et al., 2010].

III. Research Questions

After defining the problem and the robot, the following research questions can be posed:

1) By how much does closed-loop control increase the accuracy of the robot?
2) How accurate is the robot when in recognising doodles drawn on the writing medium?
3) Can the robot's physical properties be modelled precisely enough to make predictions about its behaviour?

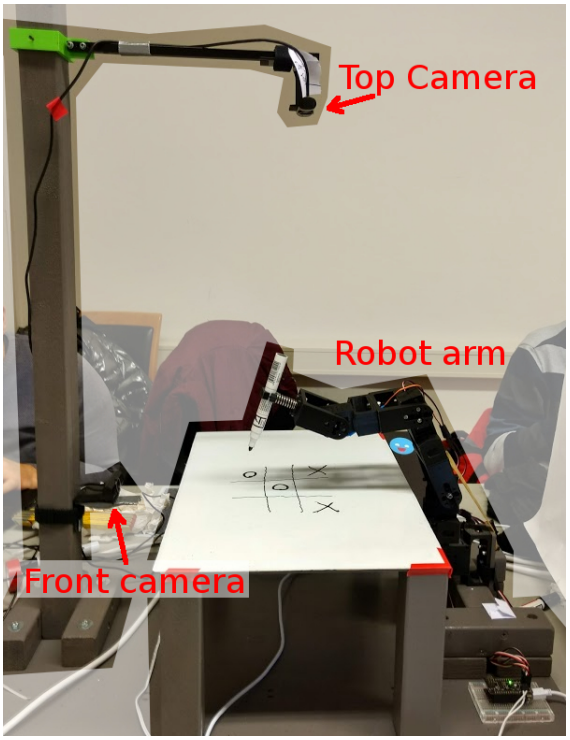

Fig. 1. Setup of the robot

IV. Robot Setup

A. Robot

The robot used in this project is a 4 degree of freedom (DOF) robot, consisting of a 1 DOF rotational (R) base linked to a 3 DOF Rotational-Rotational-Rotational (RRR) arm, where the rotational axis of the base is orthogonal to the rotational axes of the arm joints. Each joint has a certain range of motion. These ranges are:

	Base/Joint 0	Joint 1	Joint 2	Joint 3
Range	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$	$[-\frac{\pi}{2}, \frac{\pi}{2}]$

The end effector (EE) of the robot is a pen.

B. Setup

The base of the robot is fastened to a wooden structure consisting of a table and a pole. The robot is positioned in the center-front of the table, with the pole being on the other side of the table. Two cameras are fastened to the pole, one at the top of the pole, looking down on the table (and the paper), and another camera is located at a height slightly lower than the table, such that it can track the pen's height.

The robot is controlled by a microcontroller located in the base. The microcontroller is an Adafruit Feather M0 (Arduino compatible). It controls the servo motors and communicates with a PC via a serial interface. A computer can send simple movement commands to the microcontroller which it will then execute in the background. To make controlling the microcontroller/arm easier we wrote a simple python wrapper around the serial interface commands.

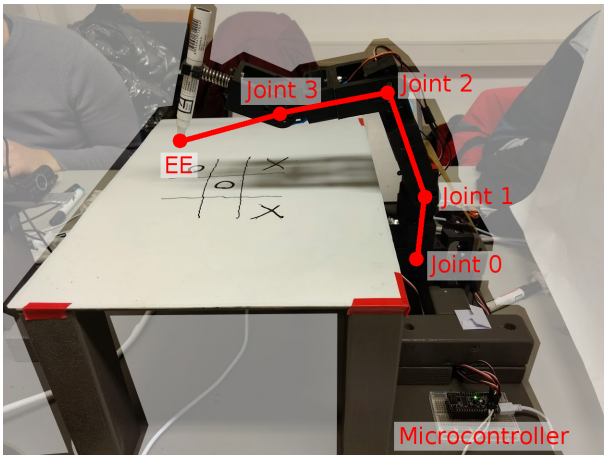

Fig. 2. Joints on the robot arm

V. Methods

A. Kinematics

Related work: Multiple inverse kinematics approaches have been developed, with two different approaches to the problem:

1) **Exact methods**: These are usually derived through geometric considerations. They are fast and exact, but become increasingly harder to derive with a rising amount of DOFs. Hand-crafted closed-form solutions become practically impossible to derive very early on. IKFast is a program which can generate analytical, closed-form solutions automatically for any robot configuration, while identifying degenerate cases and producing optimized C++ files. It is a module of OpenRAVE [ikf,].
2) **Approximate methods**: One of the most popular numerical algorithms is FABRIK due to its speed and simplicity [Aristidou and Lasenby, 2011]. FABRIK has since its publicatioon been modified to include model constraints [Aristidou et al., 2016]. Other methods like using the Jacobian [Buss, 2004], cyclic coordinate descent [Kenwright, 2012], and neural networks [Köker, 2013] have also been used.

Forward kinematics can also be performed through geometric considerations, but the by far most popular technique is using successive transformations based on Denavit-Hartenberg (DH) matrices. The DH convention standardizes the rigid transformation used to implement forward kinematics [Hartenberg and Denavit, 1955]. These rigid transformations are alternating joint and link transformations, cascading a new joint position through the whole system.

1) Inverse Kinematics: The image the robot has to draw is available as a file containing *control points* the EE has to pass through. These control points are given as spatial coordinates. Since the robot operates in angle space, these spatial coordinates first have to be mapped to joint angles. This can be achieved by employing *Inverse Kinematics*.

$$
\begin{aligned}
w_x &= x_0 + l_3 + \cos\phi \\
w_y &= x_1 + l_3 + \sin\phi \\
d &= w_x^2 + w_y^2 \\
t_1 &= \arctan\frac{w_y}{w_x} - \arctan\frac{l_2 \sin t_2}{l_1 + l_2 \cos t_2} \\
t_2 &= -\arccos\frac{d - l_1^2 - l_2^2}{2 l_1 l_2} \\
t_1' &= \frac{\pi}{2} - t_1 \\
t_2' &= -t_2 \\
t_3 &= \phi + \frac{\pi}{2} - t_1' - t_2' - \arctan\frac{e_0}{e_1}
\end{aligned}
$$

Fig. 3. Analytical solution for inverse kinematics

Analytical solution: We used a simple analytical solution based on geometrical considerations due to IKFast being difficult to use and FABRIK being cumbersome to extend to our case with constraints on joint angles.

Analytical solution: For now, assume we operate with a planar RRR robot. This significantly simplifies the analytical solution, and extending the approach to our case with another rotational base orthogonal to all other rotational axes is simple.

The analytical calculation of joint angles required to produce a given **x** is ill-posed: There are many solutions. This can be mitigated by reducing the amount of acceptable solution by restricting the orientations the EE can take. Let the EE orientation be ϕ. Then:

$$\phi = \sum_{i \in Joints} \Theta_i$$

If the solution cannot satisfy this condition, the desired EE orientation cannot be reached. Empirically, this is the case for most EE orientations. The approach we take is to start at the optimal EE orientation (with the pen being orthogonal to the paper), and then iteratively change the orientation until a solution is found. This produces empirically good results, with the EE orientation for which a solution is found deviating from the previous EE orientation by only a small amount.

The solution space shrinks further due to angle limits on each joint. The limits are specified in IV-A.

To account for the additional DOF at the base of the robot, all target positions are rotated to a common plane (in our case being the z-y-plane), and a solution for this planar case is then found by ignoring the rotational DOF about the z-axis. The required base angle is then simply the angle needed to rotate the target to the z-y-plane, with an inverted sign.

The equations determining the solution in the planar case can be found in Figure 3. They are an extension of equations given by a 2-joint system and the assumption that the orientation of the EE is fixed.

2) Forward Kinematics: Forward kinematics are needed in the simulation to reflect changes in joint angles. Forward kinematics is achieved by applying Denavit-Hartenberg transformations. The parameters for our system are:

B. Closed-Loop Control

We attempted to modify the control code to make use of closed-loop control to improve the accuracy of movements. To do this we used the position detection as discussed in V-C. the idea is to use the detected position of the EE to correct any deviations from the intended position caused by elasticity in the arm and friction with the drawing surface. The problem we observed with this approach was that the error in the position detection is larger than the error between the intended and actual position, A second problem was that the results from the position detection were very unstable, and jumped around quite erratically, which makes doing proper closed-loop control almost impossible.

However, when using our height detection only (using the front facing camera) we did manage to get quite accurate vertical movements with closed loop control. This is very limited and does not improve the overall accuracy of our arm because the height is not very important. We only care whether or not it is touching the page with sufficient force.

C. Computer Vision

1) General: Feed from two cameras and OpenCV [Itseez, 2019] are used for vision part of the program.

The front camera tracks height of the pen relative to the whiteboard. First, coloured reference point (tape) near the tip of the pen is found using colour detection. Then, distance from whiteboard to the reference point is calculated in pixels. Knowing that, length of the tape in millimeters, and distance from reference point to the tip of the pen, real world distance between pen and whiteboard is calculated.

The top camera is used to accomplish two goals. First, it provides closed-loop control with three dimensional location and rotation of final joint of the robot arm. Printed image of QR code is mounted on the joint. The camera is calibrated using OpenCV camera calibration methods [ope, 2014]. Next, features of QR code and frame from camera are found using ORB feature detector [Rublee et al., 2011]. Print of QR code is then located in camera feed using OpenCV brute force feature matching. Finally, rotation and translation are found using OpenCV Perspective-n-Point solver.

The camera is also used for finding all objects and events related to played games. Lines of Tic-tac-toe board are found using Hough Line Transform [Ballard, 1981]. Using these lines, location of nine squares in the grid is estimated. Then Harris Corner Detection [Harris and Stephens, 1988] and Hough Circle Transform are used to recognize inputs (Xs and Os) inside of grid squares.

For correct execution of game loop, it is important to know whether a hand is currently present in camera feed. Using YCbCr color space and bounds $R_{Cr} = [133\ 173]$, $R_{Cb} = [77\ 127]$ proposed by D. Chai and K. N. Ngan [Chai and Ngan, 1999] amount of skin pixels in the image is found. Then, if skin pixels make up more than 5% of total pixels in the image, hand is assumed to be present.

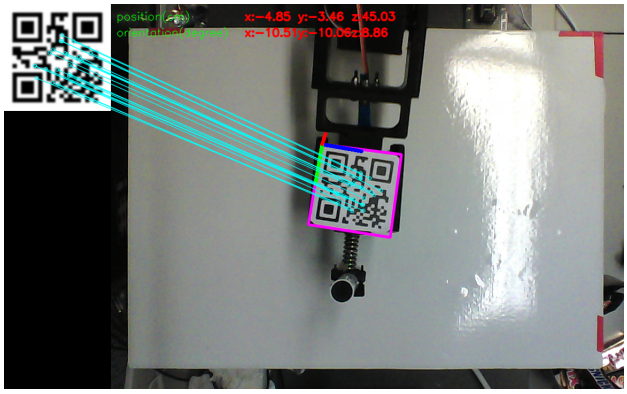

Fig. 4. Feature matching and arm position detection result.

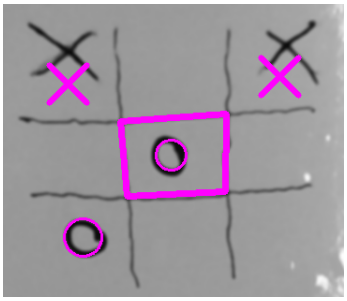

Fig. 5. Xs, Os and center square found by program drawn over real board.

2) Results: The height evaluation is good enough to work with closed-loop control, but there is some error present. The error comes from basing calculations on real world height of a piece of tape which moves together with the pen, giving incorrect calculation when it is angled. The 95% confidence interval on the error is: $[-2.82, -0.87]$mm. A solution to this problem would be using a static piece to use as a reference for real world height.

Rotation and position of final joint of arm are found (see Fig. 4), but the results are inconsistent. Even when the robot is static, the output jumps across values within few degrees and millimeters from the desired one. At this state the results are not able to be used for good closed-loop control. Further improvements are needed, since it is possible to reduce error significantly [Zheng et al., 2013].

The tic-tac-toe board detection for game logic works sufficiently (see Fig. 14) for the robot to be able to play the game against human player. The hand detection correctly classifies if a hand is in the way for the purposes of the game loop (see Fig. 6).

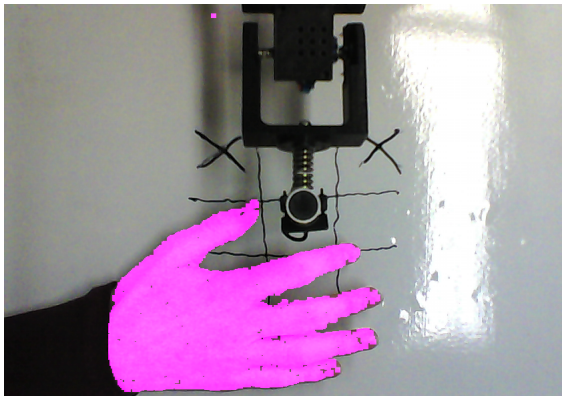

Fig. 6. Hand found by program drawn over real image.

D. AI

1) Tic Tac Toe: Minimax[Kjeldsen, 2001] was implemented for the robot to make ”intelligent” decisions in Tic Tac Toe. Minimax is a search algorithm which assumes your opponent is rational, and so will always pick the move that is worst for the minimax-player, while the minimax-player chooses the best possible move, best and worst move according to a given Heuristic.

The Heuristic takes in a board and rates it for a given player. In the current iteration, if the minimax player won the game, the board has a rating of 500, while if the opponent won the game, the board has a rating of -500. Otherwise it counts how many possible 3's in a row can be made for both players, subtracting the opponent's value from the minimax-players value - where 1 filled and 2 empty give 1 point and 2 filled and 1 empty give 2 points (negative for opponent). Values were decided arbitrarily.

2) Doodle Recognition game: To recognize doodles, the robot will utilize a classifier trained on the Google Quickdraw dataset [qda,]. As such it is limited to the classes of the dataset (345 in total). For all experiments done in conjunction with this paper, only 10 classes were used, due to hardware constraints. All experiments use 20000 datapoints from the Quickdraw dataset. Experiments were run on a Microsoft Surface Pro 4 (8GB RAM). The two main classifiers used is a linear C-support vector machine (C-SVM) [Hsu and Lin, 2002] for multiclass-classification using histograms of oriented gradients (HoG), training one-vs-many classifiers for each class[Dalal and Triggs, 2005]. C-SVM classifiers try to find a separating hyperplane in hyperspace between two sets of points, trying to maximize the distance from the nearest points. and a feed-forward neural network (NN) [**?**], where two different layer-architectures were used (see below). The NNs were trained on both raw image input and HoG. All input has been feature-scaled, fitted to the training data.

Histograms of oriented gradients are feature descriptors, i.e. a simplification of image information to its more useful parts, derived from edge intensity and direction in an image. An image is divided into cells in which gradients are computed

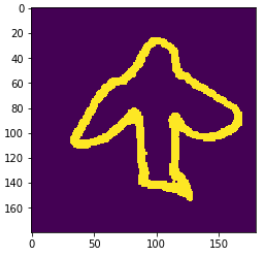

Fig. 7. Before transformation

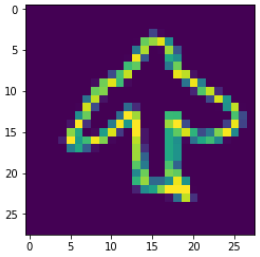

Fig. 8. After transformation

Fig. 9. Example transformation of hand-drawn doodle

from which a histogram. Cells are then normalized together in blocks, scaling values to a [0-1] range [hog,].

The Google Quickdraw is an open-source dataset, containing over 50 million doodles collected by users in which they were to draw a given category when prompted, in 20 seconds [qd,]. While the original data comes in the form of vector lines of (x,y,z), x and y being screen coordinates with a time component, the training data too comes in the form of 28x28 unsigned 8-bit integer grey-scale images.

The human-drawn doodle is drawn in an area defined by the robot - the image taken is 180x180 pixels, and is subjected to an inverse binary thresholding, and then applied a (5,5) averaging blur before being resized to 28x28 pixels, to be evaluated by the classifier.

3) Related work: The creators of Quickdraw used the collected data to create a generative RNN "... that construct stroke-based drawings of common objects." [Ha and Eck, 2017] A tutorial, linked from the dataset homepage to create an RNN for the dataset which achieves 70% accuracy. In a competition hosted on Kaggle [kag, 2018], top contenders have managed as much as 95% accuracy. However, as far as the authors are aware, there has been no investigation into the classification of doodles hand-drawn on paper using the dataset.

4) AI: Feature scaling is done by fitting on the training data, then applying it to both training and test data. [fea,] Two different machine learning algorithms, linear c-support vector machine (C-SVM) implemented with the python SKLearn package and neural network (NN). The C-SVM trains for n classes, n one vs. many classifiers, with parameters $C = 5$ (Penalty Parameter of error term). The two NNs are implemented in Keras [ker,], running on top of TensorFlow [tf,]. The first NN has a single hidden layer of 250 nodes. The second has three hidden layers, with 200, 100, and 50 nodes from input to output. There is a dropout rate of 0.5 from one layer to the next. The loss uses categorical cross-enthropy, and optimization is done with stochastic gradient descent.

Experiments were done to evaluate both classifiers, both on the Quickdraw dataset, and a collected hand-drawn dataset. For all experiments using only the Google Quickdraw dataset, data was first randomly shuffled, then split into a 16000-datapoint training set, and 4000-datapoint evaluation set. Ten classes were used in all experiments:

1) airplane
2) backpack
3) cactus
4) dog
5) ear
6) face
7) garden
8) hamburger
9) ice cream
10) jacket

When evaluating on the hand-drawn dataset, the entire 20000-datapoints were used for training instead. The first experiments are of the linear C-SVM training on different values of C with HoG-transformed data, pixel per cell (ppc) set to 4 and cells per block (cpb) set to 2. The two different NN architectures are trained on both raw pixel values and HoG-transformed data, using the same parameters for ppc and cpb. At last, the single layer NN is trained with different ppc and cpb values on the hand-drawn dataset to find the best-performing values with regards to accuracy.

5) Results:

1) **SVM Performance on Quickdraw dataset:** For different values of C, it appears that it has not much effect on the result of this dataset, staying close to a 74.1% accuracy average for all evaluations.

Test	Avg. Accuracy
SVM, C=0.5, HOG, on QD	0.749
SVM, C=1, HOG, on QD	0.746
SVM, C=2, HOG, on QD	0.731
SVM, C=3, HOG, on QD	0.723
SVM, C=4, HOG, on QD	0.735
SVM, C=5, HOG, on QD	0.746
SVM, C=6, HOG, on QD	0.746
SVM, C=7, HOG, on QD	0.744
SVM, C=8, HOG, on QD	0.742
SVM, C=9, HOG, on QD	0.746

Fig. 12. Results for different values of C when evaluating C-SVM

2) **NN Performance on Quickdraw dataset:** Both NN architectures perform similarly to each other, both on raw pixel values and HoG, with performance on raw pixel values hitting 77%, and performance on HoG hitting 84%.

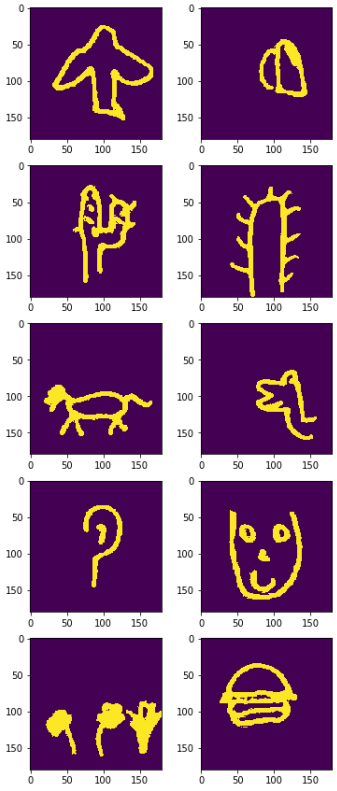

Fig. 10. Images drawn by hand

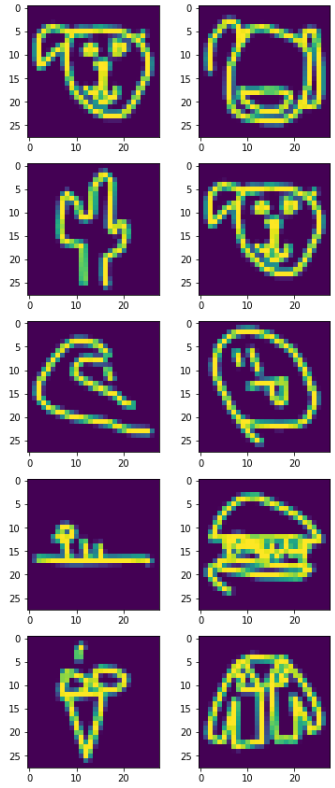

Fig. 11. Quickdraw dataset images

Test	Avg. Accuracy
NN, (250), Raw pixels, on QD	0.777
NN, (200, 100, 50), Raw pixels, on QD	0.779
NN, (200, 100, 50), HoG, on QD	0.840
NN, (250), HoG, HoG, on QD	0.843

Fig. 13. Results for training neural networks using either raw pixel input or HoG

3) **NN Performance on handdrawn data:** With raw pixel values, NN has very poor performance. Best performance was hit using HoG with ppc=7 and cpb=2, in which a 50.5% accuracy was attained.

NN (250), HoG, Hand-drawn	Avg. Accuracy
ppc = 2, cpb = 1	0.321
ppc = 3, cpb = 1	0.312
ppc = 4, cpb = 1	0.477
ppc = 5, cpb = 1	0.422
ppc = 6, cpb = 1	0.459
ppc = 7, cpb = 1	0.321
ppc = 1, cpb = 2	0.101
ppc = 2, cpb = 2	0.413
ppc = 3, cpb = 2	0.339
ppc = 4, cpb = 2	0.495
ppc = 5, cpb = 2	0.495
ppc = 6, cpb = 2	0.495
ppc = 7, cpb = 2	0.505

Fig. 14. NN performance on human hand-drawn dataset for different HoG values

A confusion matrix for the best classification on the hand-drawn data can be found below.

E. Simulation

1) Related work: Multiple programs supporting the simulation of robots exist. A notable selection of these is Gazebo, OpenRAVE , Bullet, and others [Ivaldi et al., 2014]. Although are very powerful, the extensive modeling of the robot in Unified Robot Description Format (URDF) [urd,] required by this programs is out of scope in this project. Thus, we decided to implement our own solution, which is hand-tailored to the robot we use in this project.

2) General: The implemented simulator is modular: All algorithms and methods of calculation of the simulation can be exchanged for different algorithms or methods by simply supplying other files defined by the same interface. The abstract setup of the simulator is depicted in Figure 16.

The simulator can read the same *.draw* text files as the physical robot. Additionally, the simulator applies the same algorithms as the physical robot. Interpolation algorithms which are not representable in the current state of the simulator are approximated. A physics module is used to simulate the effects of gravity.

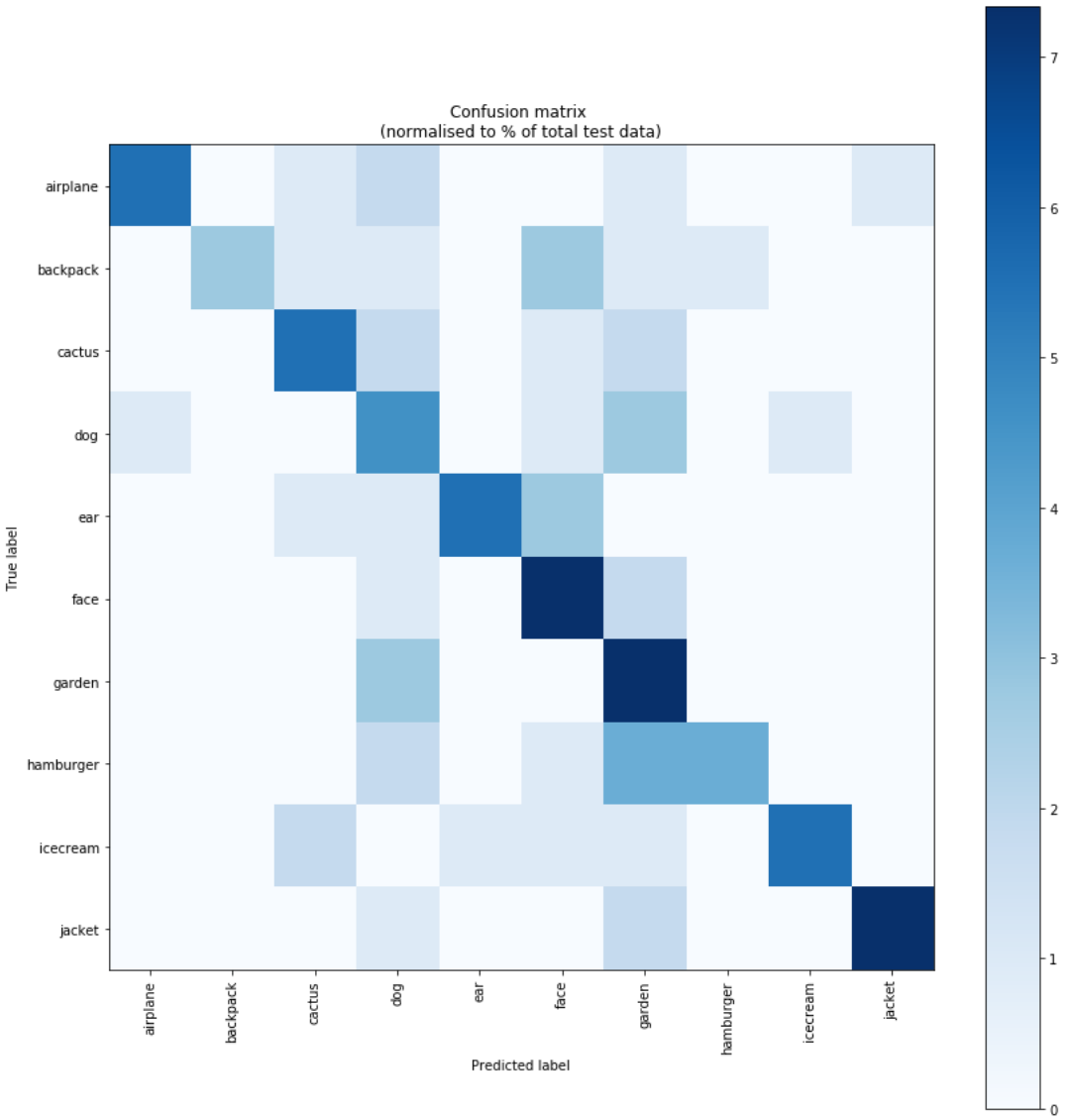

Fig. 15. Confusion matrix for neural network with HoG input (ppc=7 and cpb=2), on hand-drawn dataset

3) Physics: Simulating physical effects on the robot arm allows for a better estimation and control of inaccuracies caused by physical effects. Three effects in particular could be identified:

1) Gravitational sag of the robot arm structure.
2) Skipping of the EE due to frictional interaction between the EE and the writing medium. This has largely been resolved by switching to a writing medium with lower friction (a whiteboard).
3) Swinging of the robot arm in the x-y-plane due to non-rigidity of the base joint. This is similar to sag and can be modeled similarly.

We focused on modeling gravitational sag.

Gravity: The robot arm exhibits nontrivial amounts of sag at each joint. To model this, we assume that every joint i resembles a torsion spring with a rotational exis perpendicular to the z-axis. whose properties can be estimated using Hooke's Law:

$$\tau_i = -\kappa_i \theta_i$$

If θ_i is small, this law holds. This can indeed be assumed since the observed sagging is well below the elastic limit of the joints. Based on the assumption that this law holds, the following system of differential equations (r_j is implicitly a

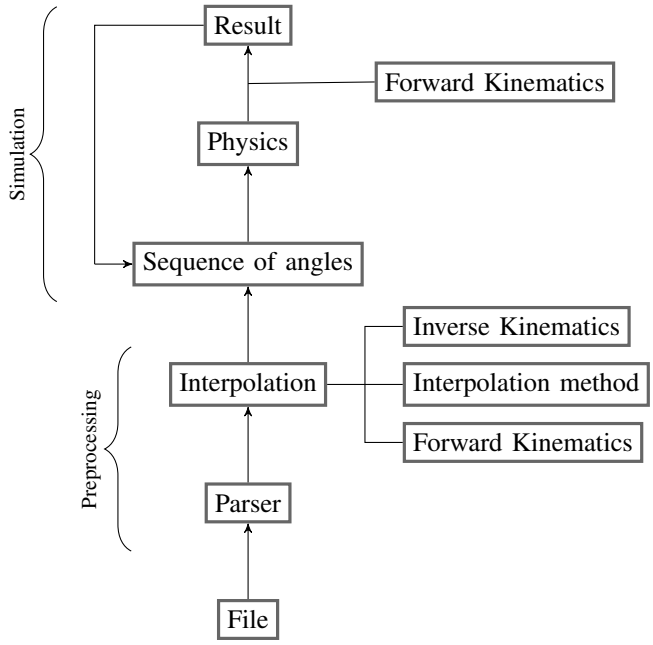

Fig. 16. Program flow and components of the simulator

function of θ_i) describe the observed effects:

$$M_i = \sum_{j=i}^{n} m_j \tag{1}$$

$$\tau_i = \sin(\psi_i) M_i |F_g| \tag{2}$$

$$\psi_i = \cos^{-1} \frac{R_i * F_g}{|R_i| * |F_g|} \tag{3}$$

$$R_i = \frac{1}{M_i} \sum_{j=i}^{n} r_j m_j \tag{4}$$

$$\theta_i = \frac{\tau_i}{\kappa_i} \tag{5}$$

θ_i is estimated analytically using Euler's method.

4) Gravity correction: Using this model for gravity it is possible to correct for the sagging effect. Note that it is possible to move the EE to a position x_G, somewhere above the writing medium, such that the EE will then be moved to a different position $\hat{x}_G$ on the surface of the writing medium due to the sagging of the robot arm structure. Then, an interpolating function can be built such that for every possible x_G on the writing medium, the function returns the corresponding $\hat{x}_G$. That way, sagging effects would be eliminated everywhere where the EE could actually affect its environment.

The needed x_G can be determined using the simulation by placing the EE at $\hat{x}_G$ and then inverting the gravitational vector to point upwards. This will place the EE exactly at x_G.

This requires a measurement of each κ_i, the spring stiffnesses. Afterwards, basinhopping optimisation [Wales and Doye, 1997] can be used to find the values of κ_i which caused the observed sag.

5) Results: We estimated each κ_i by sampling 70 points with the arm and measuring the height difference between the target position and the actual resulting position due to gravity. The full displacement could not be measured due to a too noisy sensor. The height measurement was done automatically using the height sensor we implemented. The height detection itself is noisier the farther away the pen is from the camera, so we only sampled points which were close to the camera. Even then, the noise was roughly $\pm$7mm. This taints the results significantly and the following process has to be repeated using precise height measurements. After collecting the data, basinhopping optimization, available as a library in scipy, was applied to estimate each κ_i. Since only height differences were measured, the optimization could end up in one of two minima:

1) Too low stiffness parameters: The joints sag more than they do in reality, and the EE ends up at a position closer to the base than the actual sag-induced position.
2) High stiffness parameters: The joint sag is close to the behaviour of the real robot.

In case (1), the height differences are often close to the actual height difference. This was circumvented by initializing the algorithm with very high stiffness values. Additionally, since basinhopping often takes big jumps in parameter space, the objective function Z was modified to also minimize displacement in x when applying gravity, while at the same time minimizing the height displacement prediction error Δh. This prevents it from lowering the parameters too much. Since some displacement in x is needed, the height displacement prediction error was weighted stronger. This prevents the algorithm from jumping into the minimum described in (1).

$$Z = \sum_{n} \frac{||\Delta x_n||_2 + 10|\Delta h_n|}{n}$$

Optimising using this objective function brought the parameters close enough to the true minimum, so the weight on the height displacement error was increased:

$$Z = \sum_{n} \frac{||\Delta x_n||_2 + 50|\Delta h_n|}{n}$$

After another round of optimization, the resulting values were:

	Base/Joint 1	Joint 2	Joint 3	Joint 4
Range	30.225	74.167	200.697	625.509

With a displacement in x of 0.108cm and a height displacement error of approx. 5mm on the training set. The parameters were tested on validation data consisting of 30 data points. Given a 97.5% confidence interval of $[-2.82, -0.87]$mm on the error of the height detection, the 95% joint confidence intervals for the height displacement error and height detection error are:

Height displacement error	Height detection error
[-0.583,-0.051]	[-2.82, -0.87]

The presence of the rubber band also adds complexion to the system which our method cannot model. In fact, due to the rubber band there exist different heights for each point which are stable. Afterwards, the simulation has been used to generate data points which then were used to create a gravity-correcting linear interpolation using RegularGridInterpolator in the python package Scipy.

6) Discussion: The noisiness of the height sensor severely diminishes the quality of the results. The κ_i-estimation process has to be repeated with a more precise height measurement. Nevertheless, the results achieved are a valid proof-of-concept showing the validity of the approach.

F. Other

1) Drawing Application: A drawing application has been developed which allows a user to create a drawing by placing connected lines. The lines are converted to a list of control points representing line intersections. The EE then passes through each control point. The list of control points is exported as a *.draw* file.

2) Demo: A demo video can be found under: https://youtu.be/T6it6J-Prv0

VI. Discussion

A. Research questions

We formulated three research questions:

1) By how much does closed-loop control increase the accuracy of the robot?
2) How accurate is the robot when in recognising doodles drawn on the writing medium?
3) Can the robot's physical properties be modelled precisely enough to make predictions about its behaviour?

(1): Currently the three dimensional closed loop control doesn't improve the accuracy at all, because the position detection is too noisy. However, the when only using the height detection the closed loop control manages to control the height of the EE very accurately, to within a few millimetres.

(2): Recognition currently sits at 50% for a classification problem of 10 classes. More powerful classifiers, as well as an improved paper-to-digital pipeline could lead to further improvement, and the addition of more classes.

(3): Yes, the robot can modeled precisely enough. Despite suffering from noisy data, predictions about the robot's behaviour in terms of gravitational sag can be made with reasonable accuracy, considering the data available. Better data would then presumably result in a better model.

B. Main issues and limitations

1) Doodle recognition: The training data is very clean, in the sense that all doodles have the same line width, and have been scaled to fit inside the 28x28 image. The real-world data, in contrast, has varying line-width, and each image is drawn in a slightly different position. This is why NNs perform relatively better on HoG, over raw image data - The HoG generalizes and adds robustness to noise.

2) Simulation and Modeling: The simulator is very simplistic. It is hard to extend the physics module with other physical effect. Additionally, the modeling of the robot arm was performed with noisy data. The methods used are not state-of-the-art, as more advanced methods exist.

3) Computer Vision: The camera setup is not ideal. Whiteboard is used as drawing surface, so glare is present is some lighting conditions. Tip of the marker is black which blends with the black frame of robot from the front camera point of view.

C. Future Work

1) Doodle Recognition: Multiple approaches can to improve recognition of hand-drawn doodles.

Bound improvement: Ensure that the cut-out of the doodle becomes as standardized as possible

Classifier improvement: feed-forward NNs are by no means a state of the art. a CNN approach, for example, might give better precision and improve accuracy over more classes.

Data modification: As the original data was captured in vector form, it could easily be redrawn, for example using varying line length, rotation, shift, and scale.

2) Simulation and Modeling: The robot exhibits a wobble when moving, which can be modeled in a similar way to how gravity was modeled. Damping this effect would have great effects on the drawing quality. Furthermore, repeating the joint stiffness parameter search with a precise height sensor is needed, as the system in its current state suffers from too many inaccuracies due to the high noise present in the sensor. Additionally, modeling the robot in URDF and importing it in a common robot simulation environment would improve the quality of the simulation.

3) Computer Vision: Different approaches should be explored for detecting position of the robot arm. For height, a method more reliable that just checking height of reference tape should be used. The most straightforward one would be having a visible static height reference grid in the camera feed. For calculating angles between joints, a good solution would be using an additional camera placed on side of the setup, which would look on the side of robot arm. From this camera, positions of joints and therefore angles between them could be seen clearly. State of the art approaches for Perspective-n-Point problem for use with top camera could be investigated in order to reduce the error of current method.

VII. Conclusion

The game of Tic-Tac-Toe was successfully implemented. Computer Vision aided the execution of the game successfully, but the three-dimensional position detection was too noisy to be used in closed-loop control. The robot simulation approached the real behaviour of the robot, but improvements can be made, especially in modeling physical effects on the arm. The gravity correction developed using the simulation serves as a valid proof-of-concept, but should be repeated

with better data. Doodle recognition reached barely acceptable levels of accuracy, but the corresponding game is playable. Improvements can be made in terms of selecting an appropriate architecture for doodle recognition as well as feature engineering.

APPENDIX

GLOSSARY

DOF degree of freedom. 2, 3

EE end effector. 2, 3, 7–9

R rotational. 2

RRR Rotational-Rotational-Rotational. 2

SYMBOLS

F_g The gravitational vector. 8

M_i The mass of the robot arm structure from link j up to and including the end effector. 8

R_i The center of mass of the robot arm structure from joint j up to and including the end effector. 8

Θ_i The angle of a joint i without any physical components like gravitational sag. 3

$\hat{x}_G$ The position of the end effector after moving the end effector to x_G and then letting sagging effects change the location of the end effector. 8, 10

κ_i The stiffness constant of a joint i. 7–9

ϕ The orientation of the end effector w.r.t the writing medium. 3

ψ_i The angle of attack of the gravitational vector on R_i. 8

τ_i The torque applied on a joint i. 7, 8

θ_i The change in angle of a joint i due to sag. 7, 8

m_j The mass of a link j. 8

r_j The center of mass of a link j in the base frame. 7, 8

x_G A position above the writing medium, where due to sagging effects, the end effector will be moved to a different position $\hat{x}_G$. 8, 10

REFERENCES

[hog,] Histogram of oriented gradients (explanation). https://www.learnopencv.com/histogram-of-oriented-gradients.

[fea,] Importance of feature scaling. https://scikit-learn.org/stable/auto_examples/preprocessing/plot_scaling_importance.html.

[ker,] Keras. https://keras.io/.

[ikf,] OpenRAVE — ikfast Module — OpenRAVE Documentation.

[qd,] Play quickdraw. https://quickdraw.withgoogle.com/#.

[qda,] Quickdraw dataset github. https://github.com/googlecreativelab/quickdraw-dataset.

[tf,] Tensorflow. https://www.tensorflow.org/.

[urd,] URDF - ROS Wiki.

[ope, 2014] (2014). *The OpenCV Reference Manual*. Itseez, 3.0.0 edition.

[kag, 2018] (2018). Quick, draw! doodle recognition challenge. https://www.kaggle.com/c/quickdraw-doodle-recognition.

[Aristidou et al., 2016] Aristidou, A., Chrysanthou, Y., and Lasenby, J. (2016). Extending FABRIK with model constraints. *Computer Animation and Virtual Worlds*, 27(1):35–57.

[Aristidou and Lasenby, 2011] Aristidou, A. and Lasenby, J. (2011). FABRIK: A fast, iterative solver for the Inverse Kinematics problem. *Graphical Models*, 73(5):243–260.

[Ballard, 1981] Ballard, D. (1981). Generalizing the hough transform to detect arbitrary shapes. *Pattern Recognition*, 13(2):111–122.

[Barbu et al., 2010] Barbu, A., Narayanaswamy, S., and Siskind, J. M. (2010). Learning physically-instantiated game play through visual observation. In *2010 IEEE International Conference on Robotics and Automation*, pages 1879–1886. IEEE.

[Buss, 2004] Buss, S. (2004). Introduction to inverse kinematics with Jacobian transpose, pseudoinverse and damped least squares methods. *IEEE Transactions in Robotics and Automation*, 17.

[Chai and Ngan, 1999] Chai, D. and Ngan, K. (1999). Face segmentation using skin-color map in videophone applications. *IEEE Transactions on Circuits and Systems for Video Technology*, 9(4):551–564.

[Dalal and Triggs, 2005] Dalal, N. and Triggs, B. (2005). Histograms of oriented gradients for human detection. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 1, pages 886–893 vol. 1.

[Ha and Eck, 2017] Ha, D. and Eck, D. (2017). A neural representation of sketch drawings. *CoRR*, abs/1704.03477.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. *Procedings of the Alvey Vision Conference 1988*.

[Hartenberg and Denavit, 1955] Hartenberg, R. and Denavit, J. (1955). A kinematic notation for lower pair mechanisms based on matrices.

[Hsu and Lin, 2002] Hsu, C.-W. and Lin, C.-J. (2002). A comparison of methods for multiclass support vector machines. *IEEE Transactions on Neural Networks*, 13(2):415–425.

[Itseez, 2019] Itseez (2019). Open source computer vision library. https://github.com/itseez/opencv.

[Ivaldi et al., 2014] Ivaldi, S., Padois, V., and Nori, F. (2014). Tools for dynamics simulation of robots: a survey based on user feedback.

[Kenwright, 2012] Kenwright, B. (2012). Inverse Kinematics – Cyclic Coordinate Descent (CCD). *Journal of Graphics Tools*, 16(4):177–217.

[Kjeldsen, 2001] Kjeldsen, T. H. (2001). John von neumann's conception of the minimax theorem: A journey through different mathematical contexts. *Archive of History of Exact Sciences*, 56.

[Köker, 2013] Köker, R. (2013). A genetic algorithm approach to a neural-network-based inverse kinematics solution of robotic manipulators based on error minimization. *Information Sciences*, 222:528–543.

[M. and VIEIDER, 2017] M., M. and VIEIDER, G. V. (2017). 'drawing robotic arm', dissertation.

[Megalingam et al.,] Megalingam, R. K., Raagul, S., Dileep, S., Sathi, S. R., Pula, B. T., Vishnu, S., Sasikumar, V., and Gupta, U. Design, implementation and analysis of a low cost drawing bot for educational purpose. *International Journal of Pure and Applied Mathematics, Special Issue*.

[Rublee et al., 2011] Rublee, E., Rabaud, V., Konolige, K., and Bradski, G. (2011). Orb: An efficient alternative to sift or surf. *2011 International Conference on Computer Vision*.

[Sarker, 2015] Sarker, S. (2015). Wizard chess: An autonomous chess playing robot. In *2015 IEEE International WIE Conference on Electrical and Computer Engineering (WIECON-ECE)*, pages 475–478. IEEE.

[Srikaew et al., 1998] Srikaew, A., Cambron, M., Northrup, S., A. Peters Ii, R., Wilkes, D., and Kawamura, K. (1998). Humanoid drawing robot.

[Tresset and Leymarie, 2013] Tresset, P. and Leymarie, F. (2013). Portrait drawing by paul the robot. *Computers Graphics*.

[Wales and Doye, 1997] Wales, D. J. and Doye, J. P. K. (1997). Global optimization by basin-hopping and the lowest energy structures of lennard-jones clusters containing up to 110 atoms. *The Journal of Physical Chemistry A*, 101(28):5111–5116.

[Zheng et al., 2013] Zheng, Y., Kuang, Y., Sugimoto, S., Astrom, K., and Okutomi, M. (2013). Revisiting the pnp problem: A fast, general and optimal solution. *2013 IEEE International Conference on Computer Vision*.