The Knowledge Engineering Review, Vol. 34, e27, 1 of 17. © Cambridge University Press, 2019
doi:10.1017/S0269888919000213

Dr. Eureka: a humanoid robot manipulation case study

LIN YU-REN, GUILHERME HENRIQUE GALELLI CHRISTMANN, RICARDO BEDIN GRANDO, RODRIGO DA SILVA GUERRA, and JACKY BALTES

106, Heping E. Road, Sec. 1, Taipei 10610, Taiwan
e-mails: danny40316@gmail.com, guichristmann@gmail.com, ricardo.grando@ecomp.ufsm.br, rodrigo.guerra@ntnu.edu.tw, jacky.baltes@ntnu.edu.tw

Abstract

To this day, manipulation still stands as one of the hardest challenges in robotics. In this work, we examine the board game Dr. Eureka as a benchmark to encourage further development in the field. The game consists of a race to solve a manipulation puzzle: reordering colored balls in transparent tubes, in which the solution requires planning, dexterity and agility. In this work, we present a robot (Tactical Hazardous Operations Robot 3) that can solve this problem, nicely integrating several classical and state-of-the-art techniques. We represent the puzzle states as graph and solve it as a shortest path problem, in addition to applying computer vision combined with precise motions to perform the manipulation. In this paper, we also present a customized implementation of YOLO (called YOLO-Dr. Eureka) and we implement an original neural network (NN)-based incremental solution to the inverse kinematics problem. We show that this NN outperforms the inverse of the Jacobian method for large step sizes. Albeit requiring more computation per control cycle, the larger steps allow for much larger movements per cycle. To evaluate the experiment, we perform trials against a human using the same set of initial conditions.

1 Introduction

Real-world robots can aid humans in highly repetitive tasks, or even completely replace them, for example, in a production line. However, most of the environments with which humans interact daily are adapted to our own needs and physical anatomy. A large amount of the tasks that we perform require very fine motor control. Humans acquire this kind of advanced dexterity in a very young age and go on to perform feats of amazing fine control, without hesitation or apparent effort.

The willingness to refine the precision and speed of our motions generates an interest in their benchmarking. We can easily notice this pattern in children, who repetitively challenge their peers to see who can jump the highest, run the fastest, etc. These kinds of benchmarks can be world-class challenges, like the Olympics, with professional athletes pushing human performance to its ultimate limits. We can also notice this in more routine environments, in child's play or video and board games. Simpler environments that don't require much physical labor, such as chess or similar board games, still require very precise manipulation of small objects on a surface.

Examples of computers artificial intelligence (AI) programs outperforming humans can be traced back to 1997, when the IBM's Deep Blue first overcame the world chess champion Kasparov (Goodman & Keene, 1977). Recently in 2015, AlphaGo from Google's DeepMind lab beats the top players in Go (Silver *et al.*, 2016). We have also seen development in agents that can play video games, such as OpenAI Five (OpenAI, 2018) that is also able to win against top players in the video game Dota 2[1]. However, we still need to prove that AI agents can reliably beat humans in the physical world, under the same conditions.

[1] http://www.dota2.com/

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press



Figure 1 Box Art of Dr. Eureka[2]



Figure 2 All elements of the game[3]

Some works in the direction of developing robots that can play against other humans in real-world games that involve physical manipulation have been done in the past. Kroger *et al.* (2008) used an industrial manipulator to play Jenga, a game that requires precise control and tactile feedback. They developed an accurate force/torque control system demonstrating multisensor integration and visual servoing applied to manipulation games. Calvo-Varela *et al.* (2016) used a Softbank NAO humanoid robot to play Tic-Tac-Toe against humans on a tablet, also performing visual servoing and inverse kinematics to touch points on the screen. Most notably, RoboCup (Kitano *et al.,* 1995) and FIRA HuroCup (Baltes *et al.,* 2017) are two very large robot competitions that propose research-oriented challenges and tournaments, seeking to advance the current state of the art in humanoid robotics to a point where they could, in the future, reliably compete against humans in soccer and other Olympic sports.

In this work, we present a humanoid robot that is capable of a playing a manipulation puzzle board game called Dr. Eureka (Figure 1). The game makes use of the following objects: three transparent tubes per player, six balls of three different colors (red, green and purple) per player and a deck of cards depicting configurations of the balls inside the tubes (Figure 2). To play the game, the player draws a card. The player's goal is to reconfigure the balls in the tubes in the same configuration depicted in the

[2] https://boardgamegeek.com/boardgame/181345/dr-eureka
[3] http://mypen.org.za/images/dr-eureka.jpg

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press



Figure 3 THORMANG3: Tactical Hazardous Operations Robot 3

card. The player is not allowed to directly touch the balls and can only move them by pouring from one tube to another. The quickest player to achieve the goal configuration wins.

In order to complete this game, the humanoid robot needs to perform four subtasks. (1) The robot has to be able to detect the card and understand the goal configuration depicted. Here, we assume the game always starts with the balls in the tubes in the same configuration, that is, with two balls of same color in each tube. (2) The robot employs a search algorithm to find the shortest path to the goal solution, that is, the least number of movements to reach the goal configuration. Then, (3) the robot needs to be able to locate the tubes in the world, with its camera. Finally, (4) the robot needs to physically pick and place the tubes and perform the pouring motions.

2 System overview

The robot used in this work was the humanoid THORMANG3 (Tactical Hazardous Operations Robot 3) shown in Figure 3. THORMANG3 is a complete humanoid bipedal platform sold by the South Korean manufacturer ROBOTIS[4]. It has 29 degrees of freedom (DoF), two embedded computers (Intel NUCs[5]), a camera and several other sensors. Table 1 presents the specifications of the robot, and Figure 4 shows a complete diagram of its interfaces.

In order to make communication possible between the actuators, sensors and computers, we use ROS (Robot Operating System). ROS is a pseudo operating system that allows distributed interfacing of sensors, actuators and algorithms by means of a publish/subscribe communication scheme, integrated primarily on Linux.

Figure 5 presents a diagram depicting the ROS architecture. This diagram is simplified for legibility. The core management of this system is done inside the *Dr. Eureka Main* node. The node is divided into three parts. The first part takes care of the card detection component of the task: it reads an RGB image stream from the camera node and performs image recognition to determine the configuration of the balls on the presented card. The second part is the sequence planner: using Dijkstra's algorithm (Dijkstra, 1959), the robot can find the solution for the shortest sequence of pouring motions to achieve the goal configuration. The third part is locating the tubes in the camera image, so that the robot can pick them up. The solver just decides the sequence of motions to execute (the sequence includes pickup, placing and

[4] http://www.robotis.us/
[5] https://www.intel.com/content/www/us/en/products/boards-kits/nuc.html

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

Table 1 Hardware specifications of THORMANG3.

Degrees of freedom	**29**
Actuator	200 W × 10/100 W × 11/20 W × 8
Computer	Intel NUC (i5 Processor) – 8GB RAM DDR4 × 2
Wireless router	DLink DIR-806A
Camera	Logitech C920 HD Camera
LiDAR	Hokuyo UTM-30LX-EW
F/T	ATi Mini58-SI-2800-120 x 2
IMU	MicroStrain 3DM-GX4-25
Battery	22 V, 22 000 mA – 18.5 V, 11 000 mA
Height	137.5 cm
Weight	42 kg

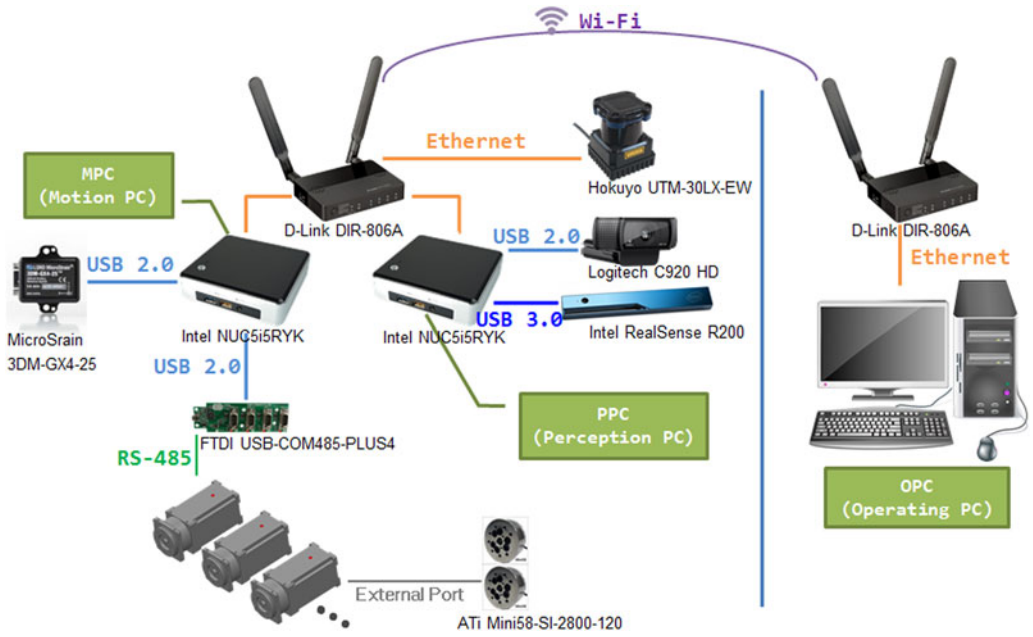


Figure 4 THORMANG3's Hardware and Devices Architecture Diagram[6]

pouring actions). Having determined the sequence, the Dr. Eureka Main node sends the planned strategy to the *Control Engine* node. The *Control Engine* node takes care of the motion management of the robot. It can read the current values of a designated joint. *Control Engine* node can also set a new position for each of the joints. For picking and placing the tubes, the Control Engine solves the inverse kinematics using an original hybrid approach that switches between using a neural network (NN) for larger steps (when the gripper is far from the goal) and using the method of the inverse of the Jacobian when the gripper is closer to the goal. For pouring the ball from one tube into another, the Control Engine node uses pre-recorded key-framed motions.

3 Design and implementation

In this section, we describe the three most important components of our system. The first part is image recognition. We combine traditional image processing to find cards and tubes and modern NN object

[6] THORMANG3 e-Manual - http://emanual.robotis.com/docs/en/platform/thormang3/introduction/

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

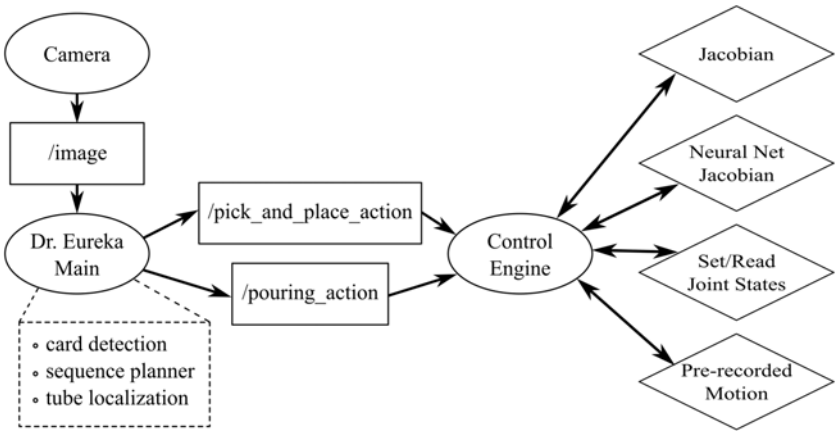


Figure 5 Diagram showing the Robot Operating System architecutre for performing the Dr. Eureka game. Ellipses represent nodes, rectangles represent topics and diamonds represent services

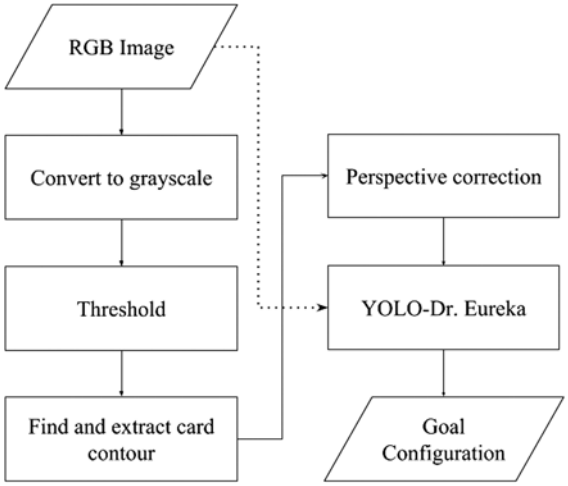


Figure 6 Flow chart for retrieving the goal configuration of the presented card

detection and classification to recognize the configuration of the balls in the card. The second part is the action sequence planning, reformulating the problem in graphs and using Dijkstra's search algorithm to find the shortest path to the goal. The third and final part is manipulation, which includes the execution of the pickup and pouring motions.

3.1 Card recognition

In this subsection, we will describe the image processing and computer vision techniques used to transform the original image from the camera and acquire the ball configuration information from the card. The diagram presented in Figure 6 shows the complete strategy for card detection.

The card recognition combines traditional techniques (for card localization) and modern NN architectures (for card identification). We designed the system in such a way that the card does not need to be placed in a specific position in front of the robot. Since the card has easily detectable features (a black background and white border), we chose traditional image processing algorithms for the task of locating the card. These features are not easily influenced by light variations. Traditional image processing techniques, such as contour detection, are usually very good at detecting this kind of low-level features without any training data and at a high computational efficiency. We use functions implemented in OpenCV to perform these transformations (Bradski, 2000).

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

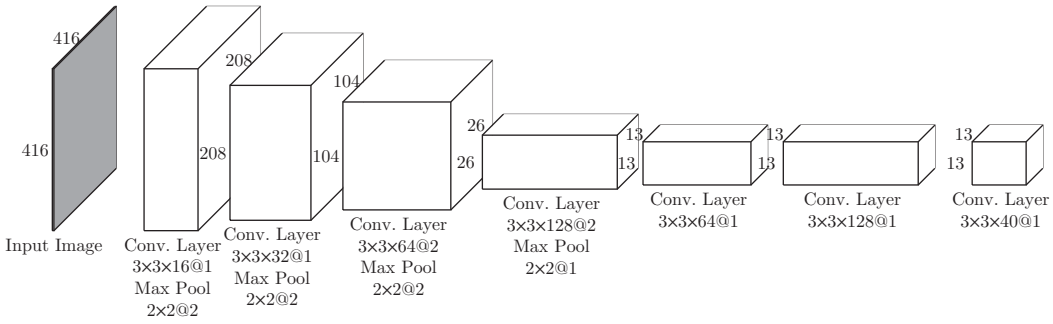


Figure 7 Our proposed custom architecture YOLO-Dr. Eureka. The output layer decodes the bounding boxes

First, we convert the RGB color image to a grayscale image. Since, for locating the card, we are only looking for the black background and white border, we don't care about color information yet. Using a threshold operation, we convert the image to binary. With the binary image, we can find all the contours. The camera is always placed at about the same height and angle from the table, so we can use the perimeter and area of the contours to find which ones are more likely to belong to the card. Due to the presence of noise in the image, we may end up detecting several vertices for the contour of the card. We use polynomial approximation to cut this number down to just four. Using these vertices, we perform an affine transformation to rectify the perspective. At this point, we have a cropped image of the presented card, with the perspective corrected. To determine the configuration shown on the card, we feed this image to a custom-designed YOLO NN model.

YOLO (Redmon *et al.,* 2016) is a very popular NN model for Object Detection in real time, outperforming other models like Fast R-CNN Girshick (2015). In the YOLO model, the candidate box and classifier are combined into a single network. Since in this application, YOLO is being applied to a very simple image recognition task, and we would like to run the model on a CPU, we trained a custom model of YOLO, greatly reducing the number of parameters, but still achieving the same performance for this specific task. It is known that the shallow layers in CNNs respond to edge and color (Zeiler & Fergus, 2013), while the middle layers respond to texture and more complex patterns. The latter layers can respond to specific abstract features, like faces, car wheels, etc. So, reducing the number of parameters for this kind of task should not affect the performance of the network too much.

We called this custom architecture *YOLO-Dr. Eureka*, specifically designed for detecting the illustration of the color balls in the rectified card image. For this task, edge, color and texture features are more important than high-level and abstract features. In terms of width, we can also use much smaller filters. In terms of deepness, we kept the dimension of input the same, so we could have the same general architecture and output dimension. We use pooling layers to reduce the dimensions of the input image, in the same manner as the original YOLOV2-Tiny (Redmon & Farhadi, 2017).

Figure 7 presents our proposed architecture. One of our modifications is in the third and fourth layers, where the CNN filter is applied with a stride of 2, in order to further reduce the size of the network and make inference faster. The output layer dimension is $13 \times 13 \times 4$, which can then be decoded into x and y coordinates for the bounding boxes, along with the confidence score and class label. The number of parameters for the original YOLOV2-Tiny is around 11 million. Meanwhile, our proposed model has approximately 250 thousand parameters, a reduction by a factor of 43.

Putting the acquired image of the card through the network, we can retrieve the x and y coordinates and the color class for each of the balls depicted in the card image. The NN output gives no information about how these balls are arranged in the tubes in the card illustration. To determine the configuration of the balls inside the tubes, we need to process the ball coordinates information. We can't directly decode the configuration information from the detected x and y coordinates because the spacing of the drawn tubes can vary from card to card (see Figure 8). To solve this, we can note two cases for the configurations: (1) either none of the tubes are empty or (2) a single tube is empty. To determine the configuration then, we perform K-Means clustering (Lloyd, 1982) on the x coordinates of the detected balls. We assume there

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

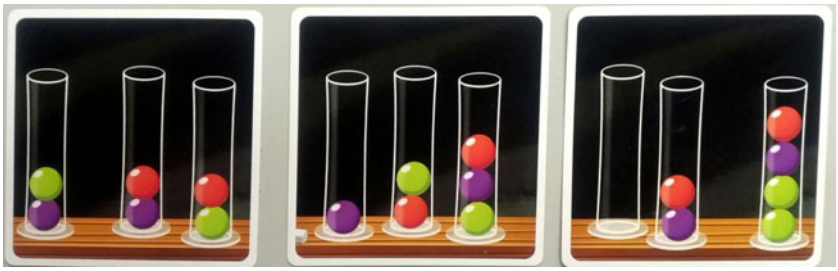


Figure 8 Spacing between the tubes varies from card to card

are three clusters in the image and then calculate the center position for each. In the case when one tube is empty, there are actually only two clusters, so by running K-Means assuming three clusters, results in two of the clusters matching the same tube. Then, to detect the empty tube cases, we test if the distance between the center positions of any two clusters is less than a very low threshold, and if so, then the two clusters are grouped into one. Otherwise, none of the tubes shown are empty. After this, we still need to know the relative positions of the tubes in the card, that is, left, center or right. For the second case, we used conditional judgment to find out the position of the empty tube.

3.2 Tube localization

For picking and placing the tubes, we needed to relate the coordinates of detected objects in the image to their corresponding 3D world coordinates in the robot's frame of reference. The first step was to calibrate the camera intrinsic parameters and distortion coefficients. This was performed using a *ChArUco* board and OpenCV's `calibrateCameraCharuco()`. After calibration, we can get the position of an *ArUco* marker relative to the camera's frame of reference, in meters. We use the robot's forward kinematics to transform the coordinates from the camera's frame of reference to the robot's frame of reference. When trying to move the gripper to a given position detected by the camera, we noticed the computed coordinates did not match exactly the robot's gripper position (computed using the robot's forward kinematics, using joint encoder information). We believe this difference is due to mechanical tolerances. To compensate for these small differences, we calibrated each arm with an affine 3D transformation mapping the computed coordinates from the camera (in the robot's frame) to the gripper coordinates (computed from joint encoder information). The affine transform was estimated using five reference points.

Since the tubes are transparent, they are very difficult to detect using standard machine vision techniques. So, we decided to add black tape to the bottom of each tube—this is the only modification that we did in the original pieces of the game. We designed a large white sheet of paper with four ArUco markers in its corners. The tubes were then placed on top of this sheet of paper. With the entire sheet of paper in the view of the robot's camera, a rectangular area (defined using corners of the ArUco markers) was rectified to remove the perspective distortion (see Figure 10, left). In this image, the black bottom of the tubes is clearly visible, due to the high contrast over the white background of the sheet of paper. The image of the bottom of the tube takes a form similar to that of a crescent moon, due to the occlusion caused by the cylinder of the tube. We used Hough transform and conditional programming to detect the larger semi-circles defined by the outer diameter of each tube (see Figure 10, right). The flow chart of the whole process is shown in Figure 9.

3.3 Shortest path problem

Next, we will discuss all the possible permutations for the configuration of the balls in the tubes, that is, the search space. In the Dr. Eureka game, we have three tubes and six balls. The balls are divided into sets of two same colored balls, red, green and purple. The maximum number of balls in a single tube is four, and one tube is allowed to be empty.

To understand the total number of possible states, we divide the problem into two parts, namely: (1) permutations of number of balls in each tube (not considering their colors) and (2) permutations of the

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

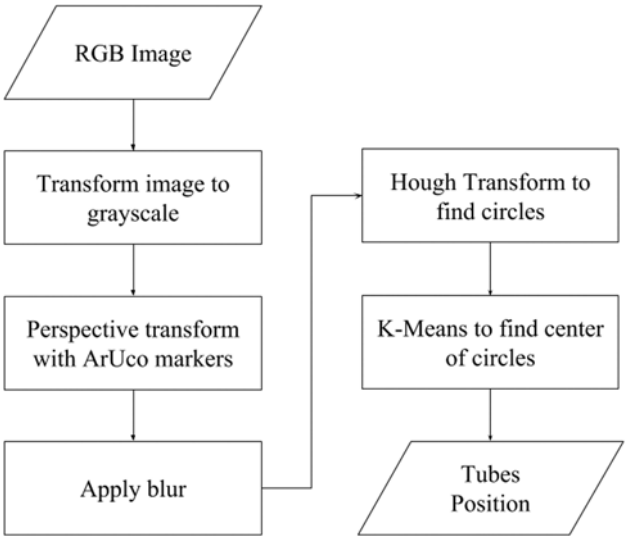


Figure 9 Flow chart for the computer vision algorithm to determine the tubes' position on the table in front of the robot

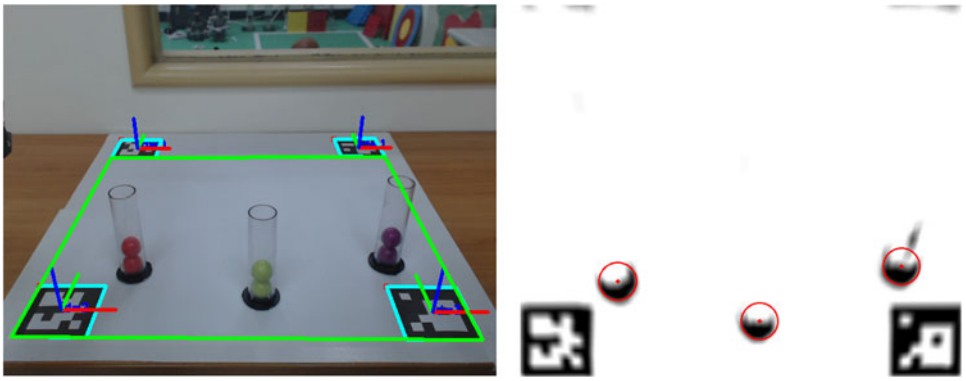


Figure 10 On the left, the sheet of paper used as a reference for detecting the positions of the tubes. On the right, the detection of the bottom of the tubes using Hough transform

order of the colors of the six balls (not considering how many balls are in each tube). For the first case, we note that for any given card, a total of six balls must be placed in the three tubes. Not considering the colors of the balls, there are a total of 19 ways in which these balls can be arranged in the three tubes. This comes from five possible configurations as follows:

1. All tubes have 2 balls each (1 case): (2,2,2)
2. One tube has 3 balls, one has 2 and one has 1 (6 cases): (3,2,1), (3,1,2), (2,3,1), (1,3,2), (2,1,3), (1,2,3)
3. Two tubes have 3 balls and one has none (3 cases): (3,3,0), (3,0,3), (0,3,3)
4. One tube has 4 balls, one has 2 balls and one has none (6 cases): (4,2,0), (4,0,2), (2,4,0), (0,4,2), (0,2,4), (2,0,4)
5. One tube has 4 balls and the other two tubes have 1 ball each (3 cases): (4,1,1), (1,4,1), (1,1,4)

It is interesting to point that these 19 possible arrangements represent only a subset of all the possible permutations of balls and spaces, since the spaces are always on top due to gravity (i.e. a configuration with a space between two balls is not possible).

Now, for the second case, not considering the arrangement of how many balls in each of the tubes, we find the number of permutations for ordering 6 balls of 2 different colors to be $\frac{6!}{2!2!2!} = 90$. Finally, we

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

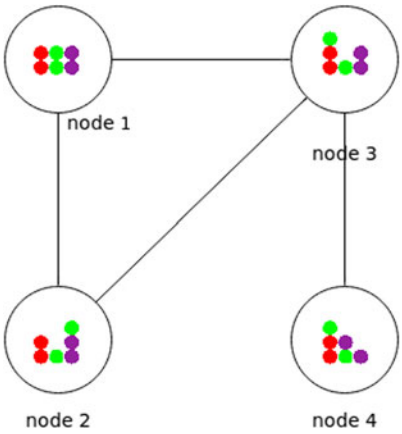


Figure 11 Small example illustrating the organization of the graph. Here, we define that node 1 represents the initial configuration and that node 4 is the goal node, while 2 and 3 are intermediate nodes

combine both cases by multiplying the 19 possible arrangements of balls in the tubes with the 90 possible ordering of the colors, determining that the Dr. Eureka game has a total of 1710 possible permutations.

We can represent each permutation as a node in a graph. In the game, we always start at the node that represents two red balls in the left tube, two green balls in the center tube and two purple balls in the right tube. We need to move to a goal node, which represents the configuration from the drawn card. When we take a pouring action, moving a ball from one tube to another, we change the configuration of the balls and tubes, so we move to a new node that represents this new configuration. This movement from one node to another is possible only when one configuration can be transformed into another by removing one ball from the top of one of the tubes and placing it to the top of another tube. Our goal is to determine the shortest path from the starting node to the goal node only following allowed transitions.

In Figure 11, we illustrate a subset of the search space. In this example, we consider node 4 to be the goal node. Node 1 is the initial configuration. To move from one node i to a neighbor node j, following the allowed transitions, we determine a cost with the following form: $C(i, j) = 1$. For example, if we want to move from Node 1 to Node 4, there are two paths we can choose. One path is: Node 1 $\rightarrow$ Node 2 $\rightarrow$ Node 3 $\rightarrow$ Node 4. This path arrives at the goal node with a total cost of 3. The second path is: Node 1 $\rightarrow$ Node 3 $\rightarrow$ Node 4. This path has a total cost of 2. The second is the shortest path, the path we want to find from the initial node to any goal node.

Knowing the graph with all possible nodes and allowed transitions (i.e. the search space), starting from the initial node (fixed for this work) and given the goal node (detected from the card), we used Dijkstra's algorithm to find the shortest path, finding the shortest sequence of movements to arrive to the goal configuration.

3.4 Visual servoing

In Section 3.2, we discussed the computer vision techniques that we used to retrieve each of the tubes' coordinates relative to the robot's frame. In order to actually pick up the tubes, we now need to move the gripper of the robot to the destination coordinates. One very popular technique for solving this problem of inverse kinematics is to iteratively compute the (pseudo)inverse of the Jacobian (Spong *et al.,* 2006) and move the robot's arm little by little, repeating the process along each pose in a trajectory. However, due to the highly nonlinear nature of the relation between the joint space and the Cartesian space, it is only possible to perform very small steps with this method. Other complications include a high computational cost and severe instability around singularities.

In a previous work of ours, we presented a NN as a potential alternative for the Jacobian method (Montenegro *et al.,* 2018). Using the same set of inputs and outputs, we proved that our method outperformed the inverse of the Jacobian for large step sizes for a simulated 3-DoF arm. The solution also proved to be very effective when moving around singularities. We also proposed a hybrid method that

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

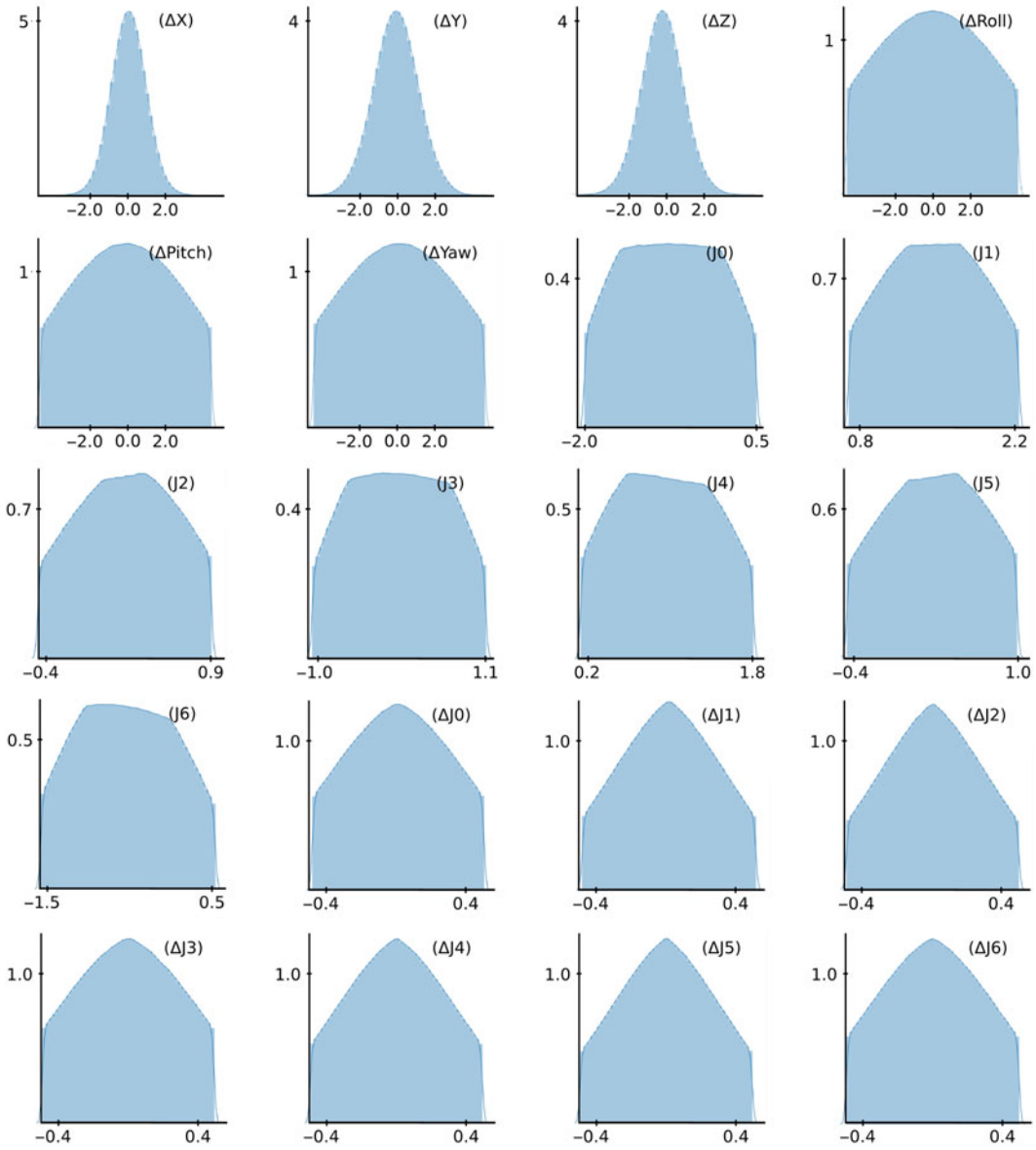


Figure 12 Distributions for each input and output of the 10 million generated samples used for training the Neural Network. *X* axis represent the range of values for the data and *Y* axis the percentage of samples at a particular *X* value. Since the data are well behaved due to the discarding of samples with large steps, we only have to normalize inputs *J*0 to *J*6.

combines both traditional Jacobian and NN approaches, reducing the number of necessary steps to reach a desired position. For this work, we implemented this hybrid approach for the first time, to perform the pickup motions for both of the 7-DoF arms of THORMANG3. This is also the first evaluation of the NN method on a real robot.

Like any NN approach, its success is highly dependent on the data it is trained with. To generate the training data, we first constrained the total workspace specifically to perform the Eureka task. We determined the limits for which each joint would be likely to still be able to pick up the tubes, with the intent of reducing the size of the search space. To generate the data, we sampled a random initial value for each joint from a uniform distribution, within the imposed angle limits. Next, we uniformly generate a random change for each joint and determine the new position of the end-effector in Cartesian space. Finally, we calculate the vector representing the change in the end-effector position, discarding samples for which we deem the moved distance too large. We generated a total of 10 million samples, whose distributions are presented in Figure 12. In that figure, we start with *Ji*, which represents the absolute joint

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

value for joint i, originally sampled from an uniform distribution, but then clipped to angle values that fall within the allowed range for all joints. After that, we generated an angular variation ΔJi, which was also sampled from an uniform distribution, and also then later clipped if the resulting absolute value $Ji + \Delta i$ fell outside the allowed range. Then, we computed the forward kinematics for both Ji and $Ji + \Delta Ji$, we found the corresponding translations ΔX, ΔY, ΔZ, $\Delta Roll$, $\Delta Pitch$ and ΔYaw in Cartesian space, and we discard all data entries for which these values represent a step larger than a given maximum threshold. These resulting distributions all look very close to Gaussian distributions.

Since the distributions end up well behaved and mostly scaled due to the sample discarding factor, we decided to only apply normalization to the current joint values input (Ji). The NN model has a total of 273 415 parameters. The architecture is composed of 7 fully connected layers, 4 layers with 256 neurons followed by 3 layers with 128 neurons. ReLU activation functions are used as well as batch normalization (Ioffe & Szegedy, 2015) between each layer. We used Adam optimizer with a fixed learning rate of 0.005 and mean squared error for the loss function. Finally, since the data distributions are well behaved and the learning gradients should be smooth, we used a very large batch size of 16 384.

3.5 Pouring actions

Every node change represents a pouring motion of one ball from one tube to another. This is achieved by first picking up the correct pair of tubes, one in each hand, and then performing either of two possible motions:

Motion 1: Pour the ball from tube in left gripper into the tube in right gripper.
Motion 2: Pour the ball from tube in right gripper into tube in left gripper.

We have now established a mapping between the results of Dijkstra's planner and robot motions. However, sometimes we can end up with two pouring actions involving the same pair of tubes, in sequence. In such cases, we do not want to waste time having the robot placing the tube back on the table and picking it up again, an action that was not needed if he held the tube from the last step. In order to avoid this, when there is a placing motion followed by a picking up motion for the same gripper and same tube, in sequence, then we delete both. For example, if the robot places back on the table the tube that was in the left gripper, and in the next step, it needs to pick it up again with the same gripper, then we simply delete the both motions. By removing these redundancies, we can improve the efficiency of the movements.

We will now perform a more detailed description of the task of pouring a ball from one tube into another. We want to note that pouring a single ball from one tube into another, starting from a tube that has two or more balls, is a very difficult feat for a robot. We use two methods to solve the task. The first one we call *lock-pouring-motion*, and it is similar to a common strategy of human players. When the human tries to pour a single ball, they often start by tilting one tube on top of the other, keeping the other tube under it so as to lock the top ball from the tilted tube between both tubes (by holding it against the edge of the destination tube). The locked ball traps the balls behind it, inside the tilted tube. Then, the human, slowly and carefully, moves the tilted tube slightly away, keeping it tilted, and keeping that top ball locked between the tubes, until the locked ball's center of gravity has passed the border of the tilted tube. Then, now with that ball completely rested on the edge of the destination tube, the human slowly straightens the tilted tube, dropping the balls that were behind the locked ball, so that they fall back inside. At the same time, only the locked ball falls into the destination tube, because its center of gravity is already outside the edge of the original tube. This motion is very efficient, with a perfect success rate. It is not only very stable but also very slow, and we can only pour one ball at a time.

The other method we call *direct-pouring-motion*, and it is based on time-delayed steps. The pouring motion is made up of two main key poses: the *ready* pose and the *pouring* pose. The motion is then a linear interpolation of both poses, with 200 intermediate poses in between. Setting a delay time between each step, we can pour either one, two or three balls in single movement, depending on the chosen delay (see Figure 13). This method is fast and able to pour more than one ball, but it does not have a perfect success rate.

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

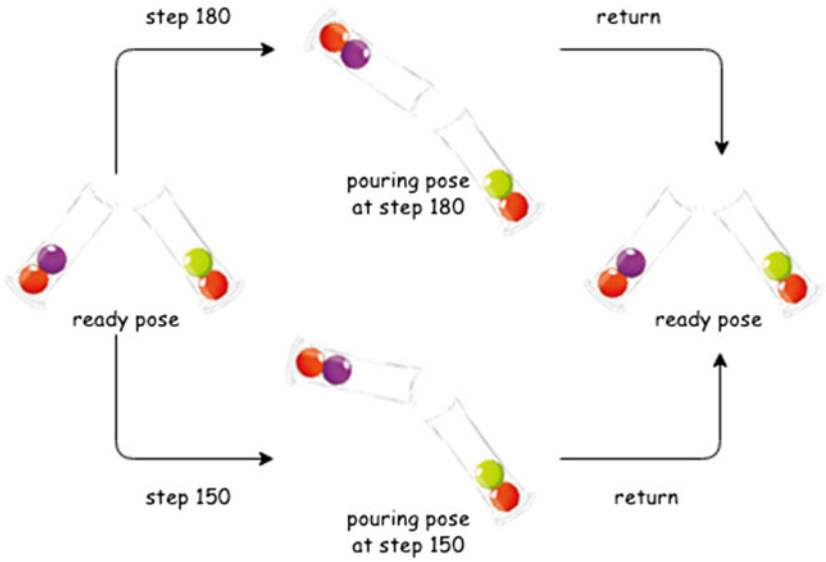


Figure 13 Flow chart of the direct-pouring-motion

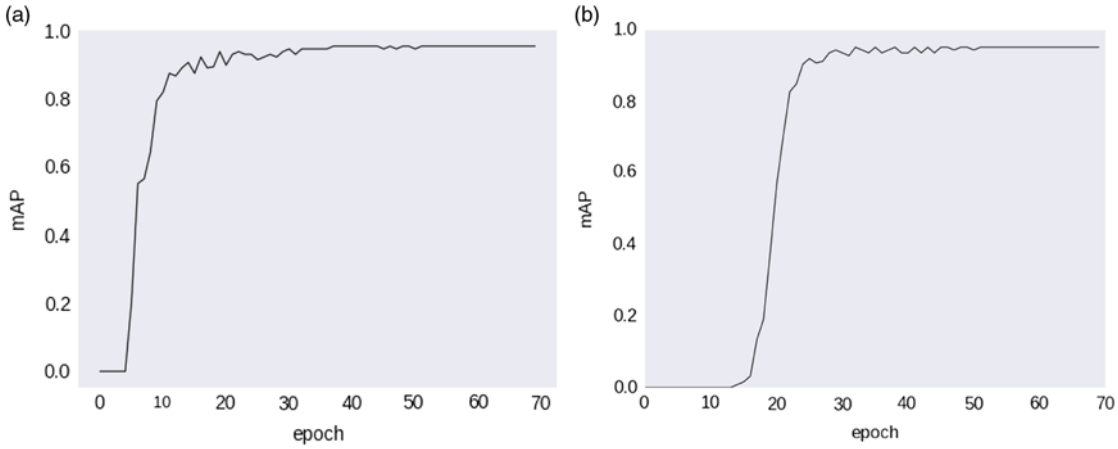


Figure 14 Mean Average Precision (mAP) over the epochs for our proposed model and the original model

4 Results

This section describes the results for each sub-component of the proposed system.

4.1 YOLO

Figure 14 shows the the validation mAP (Mean Average Precision) during training. We compare our proposed model YOLO-Dr. Eureka with the original YOLOV2-Tiny which it was based on. We can see that original model achieves a 0.9 mAP at epoch 10 and stabilizes with a mAP value of 0.952. Our proposed model needs more epochs to reach 0.9 mAP, around 25, but it still manages to get the same mAP value of 0.952. Therefore, we conclude that even though our proposed model has 43 times less parameters, we can still achieve the same level of accuracy for this specific task. Figure 15 shows a sample output for the input image of a presented card.

A big incentive to having a lower number of parameters in our proposed model is the possibility of running the model on a CPU. We measures the execution FPS (Frames-per-second) on an Intel i7-8700 3.20GHz CPU. The original YOLOV2-Tiny executed at 25 FPS, while our proposed model YOLO-Dr. Eureka achieved 50 FPS, a speedup of two times.

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

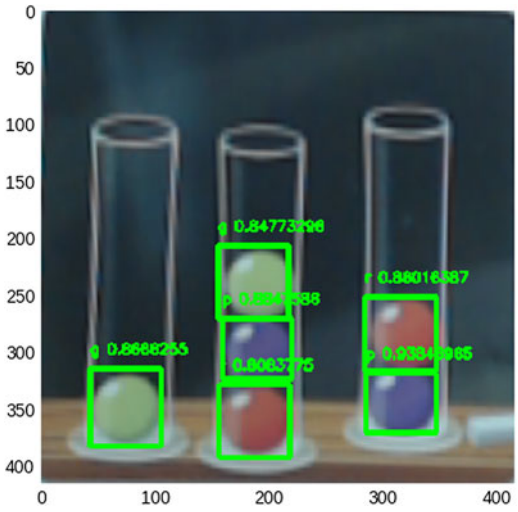


Figure 15 Sample output for our proposed model. 'G' stands for green ball, 'R' for red and 'P' for purple



Figure 16 Human performance, compared to the perfect solution (by the robot)

4.2 How does a human plan?

Out of curiosity, in this section, we explore the performance of a human for the same test cases applied to the robot. In this game, the major difference between a human and an AI agent is that humans are only capable of short-term planning, while the AI can easily handle long sequence planning optimally. We can think of the stack of balls inside the tubes as FIFO-like: the first ball to come in is the first ball to come out. If we want to pour a ball that is under another ball, we first need to remove the top ball. Due to this, the average person usually chooses the same strategy: they try to make the configuration of the balls in the bottom of the tube the same as the goal configuration, and then beginning from the bottom they check if the configuration matches the goal. This kind of planning nearsightedness is caused by the human nature of attempting to break down problems by first achieving the nearest subgoals.

Figure 16 presents the flow of the sequence of actions both for a human and our robot, for two different initial configurations, a simple (top) and a more complex one (bottom). Even for the simple initial configuration, the human has a hard time determining which moves are beneficial in the long term, while the robot can easily plan the best sequence of actions to reach the end goal. For the complex case, the human takes a lot of random actions, having even greater difficulty planning long term and approaching the problem greedily.

Figure 17 shows how the cost needed to reach the goal position increases for the human, for incrementally more complex initial conditions, departing from the optimal solution (in red). We randomly generate cases which have an optimal solution of X cost (least number of steps needed to reach goal configuration). Human performance is comparable to that of the optimal until it reaches initial configurations

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

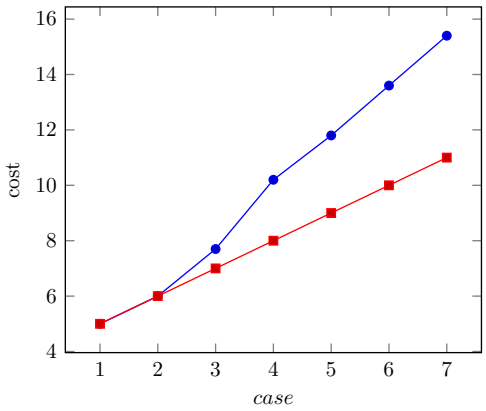


Figure 17 Comparison of cost to reach goal position for human (blue) and robot (red). Each case is a different initial condition, where the number of steps for the perfect solution is increased by 1

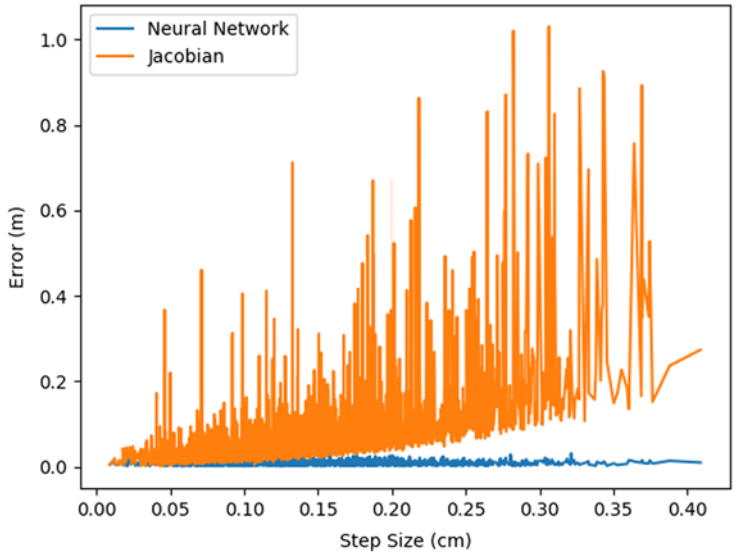


Figure 18 Error comparison between Neural Network (NN) and inverse of the Jacobian for varying step sizes. Note that the NN error remains consistent even for very large steps.

where seven or more steps are needed to reach the goal state. From then on, the human performance gets progressively worse. The cost of each presented case is the average of 10 different initial conditions, with the same number of required steps for the perfect solution. These results, again, reinforce the idea that humans are not good long-term planners.

4.3 Visual servoing

Figure 18 presents a comparison between the error of the NN model and the inverse of the Jacobian. The test was performed using the same random initial position for both methods with an increasingly larger step size. We can notice that the error for the Jacobian method grows exponentially as the step sizes become larger. On the other hand, the error for the NN stays consistently small even for distances larger than 30 cm. During the execution of the task with the real robot, we employed the previously proposed hybrid approach described in Section 3.4 on THORMANG3. When the distance to the goal position exceeded 12 cm, we used the NN to perform steps of approximately 10 cm. For smaller distances, we used the inverse of the Jacobian with steps of approximately 0.5 cm. The approach proved very stable,

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

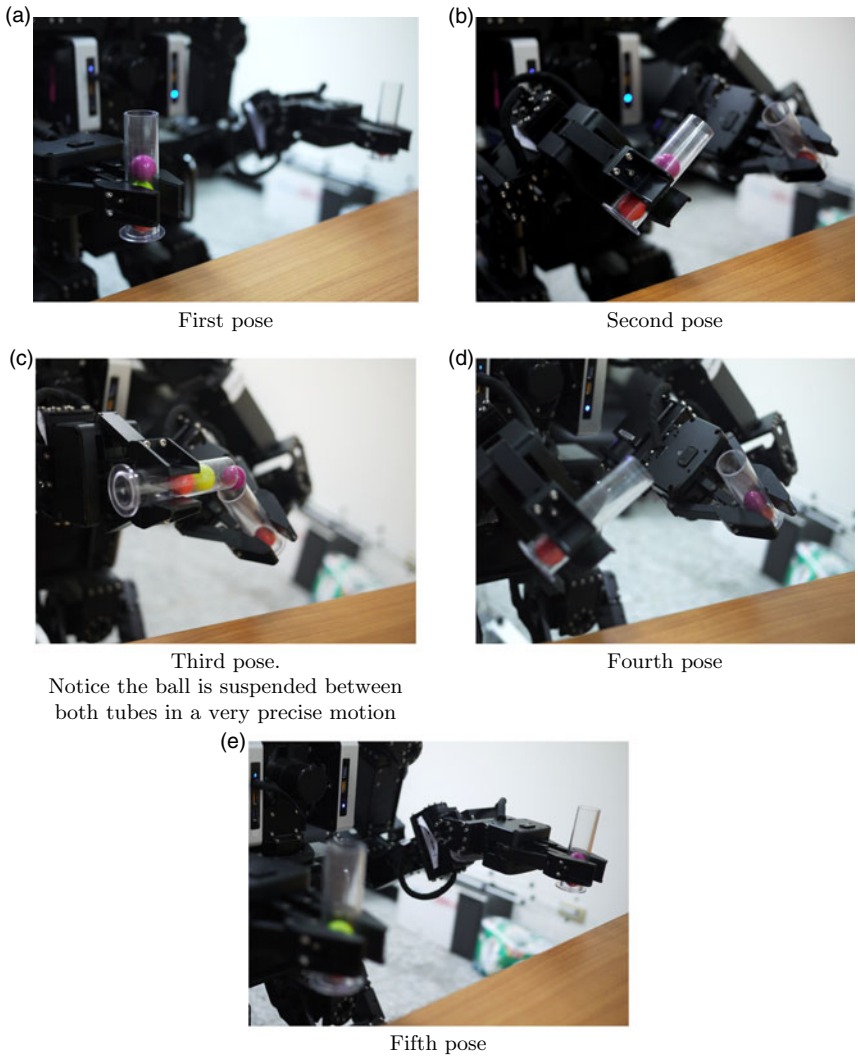


Figure 19 Sequence of poses for the Lock-Pouring-Motion motion

with the NN quickly converging to neighborhood of the goal and the inverse of the Jacobian fine tuning for reaching the goal position with precision.

4.4 Pouring actions

Figure 19 displays each key pose in the sequence of motions for performing the lock-pouring-motion. Figure 19(a) shows the initial pose, just after the pickup motion. This is the bridge pose between picking the tube and the pouring motions. Next, Figure 19(b) shows the preparation pose. The first and the second poses are distinguished because sometimes we want to do the pouring motion multiple times, avoiding redundant pickup motions. Figure 19(c) shows the lock phase. Notice the purple ball, which is locked between both tubes, in a very precise position (the ball's center of gravity is already outside the border of the original tube). Then, in Figure 19(d), we return to the preparation pose, with the balls that were behind the locked ball naturally falling back and the locked ball falling into the destination tube. Finally, in Figure 19(e), we return to the initial pose, ready to perform a pick-up or place motion next. This lock-pouring-motion has a success rate of 100%, always pouring the ball from one tube to another perfectly in all trials.

Table 2 presents the results for the direct-pour-motion for different conditions. The advantage of this method is that we can pour more than a single ball in a motion. For this experiment, the total time steps

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

Table 2. Results for direct-pour-motion for passing different numbers of balls in a single motion.

Number of balls before pouring	Number of balls to pour	Balls to leave after pouring	Return delay (ms)	Success rate
4	1	3	120	90%
4	2	2	190	90%
4	3	1	210	80%
3	1	2	130	90%
3	2	1	190	90%
2	1	1	130	100%

are 230. The delay time between each step is 5 ms. We perform 10 trials for each condition to determine the success rate. We previously determined the delays for the return motion according to the number of balls we wanted to pour. We used 120, 190 and 210 for pouring one, two or three balls, respectively. We can notice a pattern where the success rate becomes progressively worse as the return delay grows.

5 Conclusion

In this paper, we presented a humanoid robot (THORMANG3) that is capable of playing a precise manipulation game called Dr. Eureka[7]. We pose the game as a shortest path problem and employ Dijkstra's algorithm to find the optimal sequence of actions to the solution. The manipulation problem was divided into two parts: picking up the tubes (through inverse kinematics) and pouring motions (handcrafted). We solve the inverse kinematics problem using a novel NN-Jacobian hybrid. Using the same set of inputs and outputs, for distances over 12 cm, we execute 10 cm steps using the NN and 5 mm steps for smaller distances using the inverse of the Jacobian. We also show that our NN model outperforms the Jacobian at large distances. To execute the pouring motions, we use carefully designed interpolation of poses with precise timings. To detect the goal configuration in a presented card, we employ traditional image processing techniques along with a custom model of YOLOV2-Tiny. Our custom YOLO-Dr. Eureka model can reach 50 FPS when running exclusively on CPU. To localize the tubes in the real world and convert them to the robot's coordinate frame, we use Hough transform in combination with ArUco markers.

In the future, we want to investigate our NN-Jacobian method further, with a larger workspace and comparing the effects of different NN architectures in the performance. However, our main interest, right now, is in the development of dynamic pouring motions using learning algorithms as opposed to the handcrafted key-framed motions used in this work. We are exploring the use of deep reinforcement learning techniques in conjunction with dynamic movement primitives, to allow the robot to learn the pouring motions by itself.

Acknowledgement

This work was financially supported by the 'Chinese Language and Technology Center' of National Taiwan Normal University (NTNU) from The Featured Areas Research Center Program within the framework of the Higher Education Sprout Project by the Ministry of Education (MOE) in Taiwan, and Ministry of Science and Technology, Taiwan, under Grants No. MOST 108-2634-F-003-002, MOST 108-2634-F-003-003, and MOST 108-2634-F-003-004 (administered through Pervasive Artificial Intelligence Research (PAIR) Labs), as well as MOST 107-2811-E-003 -503-. We are grateful to the National Center for High-performance Computing for computer time and facilities to conduct this research.

[7] A video demonstration can be accessed at https://youtu.be/XWSNiQHQsBI.

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press

References

Baltes, J., Tu, K.-Y., Sadeghnejad, S. & Anderson, J. 2017. Hurocup: competition for multi-event humanoid robot athletes. *The Knowledge Engineering Review* **32**, E1.

Bradski, G. 2000. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*.

Calvo-Varela, L., Regueiro, C. V., Canzobre, D. S. & Iglesias, R. 2016. Development of a nao humanoid robot able to play tic-tac-toe game on a tactile tablet. In *Robot 2015: Second Iberian Robotics Conference*, 203–215, Springer.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische Mathematik* **1**(1), 269–271.

Girshick, R. B. 2015. Fast R-CNN. *CoRR*, vol. abs/1504.08083.

Goodman, D. & Keene, R. 1977. *Man Versus Machine: Kasparov Versus Deep Blue*. H3 Inc.

Ioffe, S. & Szegedy, C. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift, *arXiv preprint arXiv:1502.03167*.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I. & Osawa, E. 1995. *Robocup: The Robot World Cup Initiative*.

Kroger, T., Finkemeyer, B., Winkelbach, S., Eble, L.-O., Molkenstruck, S. & Wahl, F. M. 2008. A manipulator plays Jenga. *IEEE Robotics & Automation Magazine* **15**(3), 79–84.

Lloyd, S. 1982. Least squares quantization in PCM. *IEEE Transactions on Information Theory* **28**(2), 129–137.

Montenegro, F. J. C., Grando, R. B., Librelotto, G. R. & Guerra, R. d. S. 2018. Neural network as an alternative to the jacobian for iterative solution to inverse kinematics. In *2018 XV Latin American Robotics Symposium and IV Brazilian Robotics Symposium (LARS/SBR)*, João Pessoa, Brazil, IEEE.

OpenAI. 2018. Openai five. https://blog.openai.com/openai-five/.

Redmon, J. & Farhadi, A. 2017. Yolo9000: Better, faster, stronger. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July.

Redmon, J., Divvala, S., Girshick, R. & Farhadi, A. 2016. You only look once: Unified, real-time object detection. In *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June.

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Van Den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., *et al.* 2016. Mastering the game of go with deep neural networks and tree search. *Nature* **529**(7587), 484.

Spong, M. W., Hutchinson, S., Vidyasagar, M., *et al.* 2006. *Robot Modeling and Control*.

Zeiler, M. D. & Fergus, R. 2013. Visualizing and understanding convolutional networks, *CoRR*, vol. abs/1311.2901.

https://doi.org/10.1017/S0269888919000213 Published online by Cambridge University Press