Robotic throwing controller for accelerating a recycling line

Norman Marlier[1,2] , Olivier Brüls[1], Godefroid Dislaire[3] and Gilles Louppe[2]
[1]Department of Aerospace and Mechanical Engineering, University of Liège, Belgium
[2]Department of Electrical Engineering and Computer Science, University of Liège, Belgium
[3]Department of Architecture, Geology, Environment and Civil Engineering, University of Liège, Belgium
{norman.marlier, o.bruls, godefroid.dislaire, g.louppe}@uliege.be

***Abstract*— Recycling is a promising way to prevent the use of raw material and reduce energy consumption, air pollution and waste. However, the process of recycling has to be economically efficient in order to be adopted by industrial manufacturers. One way to achieve this goal is to improve the recycling rate. We propose a novel method to design a machine learning-based controller to improve the efficiency of a recycling line by throwing waste into buckets instead of picking and dropping them. Our proof-of-concept is demonstrated on stones, because of their simple and uniform shapes. The method enables an ABB IRB 340 robot to throw objects to buckets with an empirical success rate of 99%.**

I. INTRODUCTION

Automation and robotics are popular methods to improve the efficiency of production lines. Industrial robots perform quickly and with high accuracy simple tasks to manufacture objects. However, they suffer from several drawbacks. First, they are very task-specific and evolve in a controlled and structured environment. If the environment or the task slightly changes, they need to be reprogrammed. This mean they cannot deal with uncertainty. This lack of flexibility can be quite expensive for industrial manufacturers. Second, they cannot achieve complex tasks due to the implementation of explicit instructions in their programs. This can be problematic for future developments of industrial processes.

Machine learning techniques, especially reinforcement learning [1], are becoming more and more popular in the domain of artificial intelligence and robotics. They can achieve super-human performances on certain tasks, such as video games [2]. With these techniques, robots can interact with an unstructured environment and progressively learn how to act in a given situation. They are not preprogrammed anymore, but their programs evolve over time to perform better and better.

Recycling lines need to deal with the problem of sorting. The recycling line used in this study is as follows. Waste comes from a vibrating platform and is distributed over a conveyor. Waste passes through a set of sensors, a 1D camera, a hyperspectral camera, X-ray sensor and a LIBS, in order to determine the nature of the matter (aluminum, copper, zinc, lead, etc.) and the precise composition of alloys [3]. Moreover, the position on the conveyor and the mass are also determined. Waste is then sorted in different buckets in order to be recycled, which is the main goal of the line. Because of the large variety of waste, robots are employed to sort them. The current sorting method is a pick-and-drop operation. That is often inefficient when dealing with a high speed conveyor (>1m/s).

Contribution Our main contribution is the design of a robotic controller guided by a neural network classifier for throwing waste into buckets in order to save time and improve the recycling rate.

II. METHOD

A. Problem statement

The problem of throwing objects into buckets is formalised as the problem of finding the action a^* in state s which maximizes the probability of success of the throw. It is equivalent to minimizing the probability of failure, $P(r = 0|s, a)$, where

$$r = \begin{cases} 1 \text{ if the throw succeeds,} \\ 0 \text{ otherwise.} \end{cases}$$

The problem is an optimization problem defined as follows,

$$a^* = \min_a P(r = 0|s, a). \tag{1}$$

We define the action space $\mathcal{A} = \{(t, y, z) \in \mathbb{R}^3\}$, where the three action variables $\{t, y, z\}$ correspond to the time t to wait before opening the gripper, the horizontal displacement y and the vertical displacement z (Fig. 1).

These variables are bounded as $t \in [60, 110], y \in [0.04, 0.1], z \in [0.04, 0.09]$, where t is in $[ms]$ and y, z are in $[m]$. We also define the state space $\mathcal{S} = \{s \in \mathbb{R}|s = [-0.2, 0.2]\}$, where s is the distance of an object from the center of the conveyor, in $[m]$.

The probability $P(r|s, a)$ is unknown but can be approximated from observed data using machine learning. The resulting approximation $\hat{P}(r|s, a)$ leads to an approximate solution of the action a^*. The final problem is thus

$$\hat{a}^* = \min_a \hat{P}(r = 0|s, a). \tag{2}$$

This problem requires only a one-step prediction which simplifies strongly the computation time and the complexity. The optimization method uses the L-BFGS algorithm [4] to converge quickly to a solution. The optimization procedure is repeated 20 times by starting from different points in order to avoid getting stuck into a local minimum.

We measure the score of our approach by the empirical success rate (ESR), which is the number of successful throws over the total number of throws.

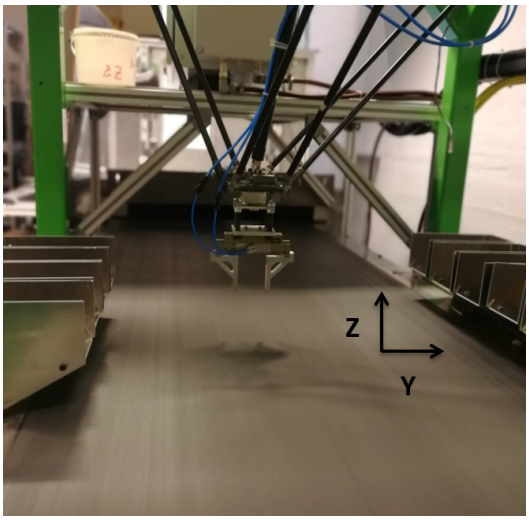


Fig. 1: The ABB robot with it pneumatic gripper on the recycling line. The buckets are located on the sides of the conveyor.

B. Model

The model used to approximate $P(r|s,a)$ is a neural network. Because r is a binary variable, this is a regular binary classification problem. The inputs of our neural network are the state and the actions, (s,a) and its output is r. We train a neural network to fit at best the relation $r = f(s,a)$ to compute $\hat{P}(r=0|s,a)$. The hyper-parameters of the model, *i.e*, the number of hidden layers and the number of neurons by layers, are chosen by cross-validation [5] with the ROC-AUC metric [6] (see Table. I). The final choice is a neural network with two hidden layers with respectively seventy and eighty neurons.

Table I: ROC-AUC obtained by cross-validation (cv=5) with different hyper-parameters.

Number of neurons	Mean score	Std score
10	0.892	0.14
30	0.905	0.12
50	0.904	0.124
70	0.907	0.118
100	0.903	0.12
10, 10	0.891	0.13
40, 30	0.9	0.097
70, 80	**0.91**	**0.081**

The choice of a neural network is also motivated by the time constraint. In fact, our small neural network is approximately an order of magnitude less time-consuming to compute the probability in contrast to ensemble trees methods, such as Extremely randomized trees [7].

The neural network was implemented with the open source library Scikit-Learn [8], as a *MLPClassifier* with Adam optimizer [9].

C. Training procedure

The training procedure is the key to quickly succeed in the task of throwing objects into buckets. The procedure needs to converge to the best action and yet to explore enough situations to generalize well. This is known as the exploration/exploitation dilemma. For practical reasons and its simplicity, the ε-greedy method [1] is chosen to solve this dilemma. This method consists in taking the best action with a probability $1-\varepsilon$ and a random action with a probability ε. The training procedure is defined as follow:

1) At step i, 50 samples are generated by the ε-greedy method and constitute the dataset $\mathcal{D}_i$.
2) 50 more samples are generated without the ε-greedy method to compute the ESR.
3) The neural network is trained by using the dataset $\mathcal{D}_i$.
4) Repeat 1-3 N times.

This process will be repeated ten times and random actions are taken to initialize the training process.

III. EXPERIMENTS

A. Simulation

The use of a simulator is a common practice in the reinforcement learning community [10, 11] because of its advantages: data are freely and quickly available, no expensive damages occur on real hardware. However, a simulator is a simplified version of the real world. A policy learnt in a simulator can fail in the real world.

Our simulator is based on several assumptions:

1) The simulator implements the equations of projectile motion with no air friction.
2) The gripper opening time is bounded between the two experimental values $80[ms]$ and $90[ms]$.
3) Velocity and acceleration are constrained due to the hardware constraints.

The simulator has two purposes. First, one can verify if the task can be achieved or not. Second, it gives an idea of how much data is needed to learn the task. It can be seen in Fig. 2 that the task is nearly perfectly learnt and only about one hundred samples are needed to learn the task. In order to choose the right value for the ε-greedy method in the real setup, four values are tested in the simulator. Because it exists few differences between these values, the value $\varepsilon = 0.1$ was chosen for the real setup.

B. Data acquisition

Learning in the simulator shows that one hundred samples of (s,a,r) are needed to achieve a good ESR, *i.e*, $\sim 90\%$. This is obviously a lower bound, because the simulator does not model the real-world complexity of the task. Because of the setup, acquiring data by hand is a tedious and error prone process. Therefore, the process was automated by using a camera to detect whether the throw is a successful by looking at the bucket. The detection method, called *Background subtraction*, compares the background, which is basically the stationary part of the image (Fig. 3b), and the foreground, which is the change in the image (Fig. 3a). We can detect if a object is in the bucket or not (Fig. 3). The algorithm is implemented in OpenCV [12] and runs on an Odroid X4 plateform.

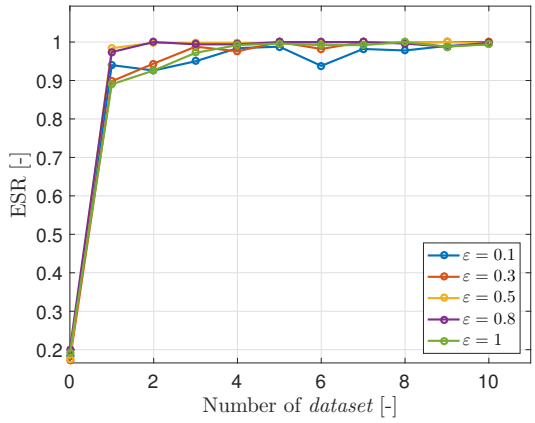
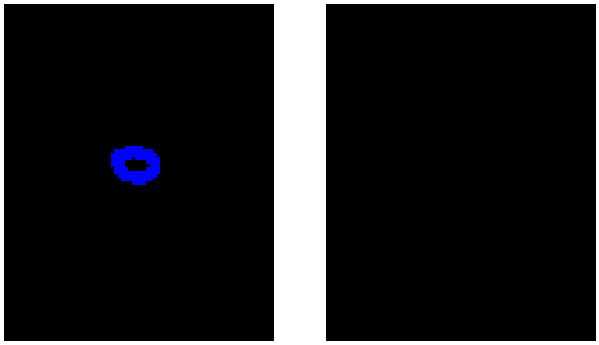


Fig. 2: ESR comparison between several values for ε-greedy policy in the simulator. One dataset corresponds to 50 samples.



(a) With an object. (b) Without.

Fig. 3: Object detection by using background subtraction method.

C. Real-life problem

Compared to the simulator, the neural network will learn more slowly in the real world setup. Indeed, as it can be seen in Fig. 4, the model needs about 250 samples to achieve a good ESR ($\sim 90\%$).

Concerning the probability of success as a function of the distance from the bucket, it can be seen in Fig. 5 that the neural network is confident along the width of the conveyor. The probability quickly drops outside the range of possible throws. But it is not relevant for this application.

A last experiment was conducted in order to compute to final score when the neural network was trained on the entire dataset. The robot throws 34 stones and succeeds in throwing all, achieving an ESR of 100%.

IV. CONCLUSION

Controllers based on machine learning techniques enable robots to achieve more and more complex tasks without explicit programming, as it is usually the case for industrial applications. Despite a very simple architecture, the neural network performs very well on the throwing task, achieving nearly an ESR of 99% with time constraints.

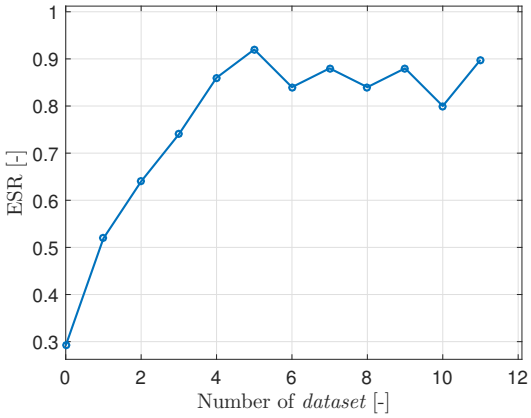


Fig. 4: ESR in real setup for $\varepsilon = 0.1$. One dataset corresponds to 50 samples.

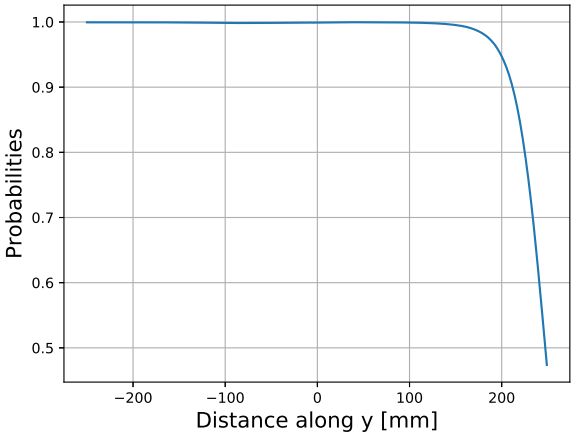


Fig. 5: The probability of success from a distance y of the bucket while taking the best action.

Furthermore, the use of a simulator was very helpful to estimate how much data is needed to learn. The limitations of the simulator were also shown: the dynamic behavior of the pneumatic gripper is very difficult to simulate.

Future work may investigate if adding more steps in the decision-making process and taking into account more sensor information, such as geometrical parameters, can improve the ESR for complex shape objects and thus get a better recycling rate.

ACKNOWLEDGEMENT

The first author would like to acknowledge the Belgian Fund for Research training in Industry and Agriculture for its financial support (FRIA grant).

REFERENCES

[1] Richard S Sutton and Andrew G Barto. Reinforcement learning: An introduction. 2011.

[2] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A Rusu, Joel Veness, Marc G Bellemare, Alex Graves, Martin Riedmiller, Andreas K Fidjeland, Georg Ostrovski, et al. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529, 2015.

[3] Pierre Barnabé, Godefroid Dislaire, Sophie Leroy, and Eric Pirard. Design and calibration of a two-camera (visible to near-infrared and short-wave infrared) hyperspectral acquisition system for the characterization of metallic alloys from the recycling industry. *Journal of Electronic Imaging*, 24(6):061115, 2015.

[4] Ciyou Zhu, Richard H Byrd, Peihuang Lu, and Jorge Nocedal. Algorithm 778: L-bfgs-b: Fortran subroutines for large-scale bound-constrained optimization. *ACM Transactions on Mathematical Software (TOMS)*, 23(4):550–560, 1997.

[5] Ron Kohavi et al. A study of cross-validation and bootstrap for accuracy estimation and model selection. In *Ijcai*, volume 14, pages 1137–1145. Montreal, Canada, 1995.

[6] Andrew P Bradley. The use of the area under the roc curve in the evaluation of machine learning algorithms. *Pattern recognition*, 30(7):1145–1159, 1997.

[7] Pierre Geurts, Damien Ernst, and Louis Wehenkel. Extremely randomized trees. *Machine learning*, 63(1):3–42, 2006.

[8] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. Scikit-learn: Machine learning in python. *Journal of machine learning research*, 12(Oct):2825–2830, 2011.

[9] Diederik P Kingma and Jimmy Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[10] Mark Cutler, Thomas J Walsh, and Jonathan P How. Real-world reinforcement learning via multifidelity simulators. *IEEE Transactions on Robotics*, 31(3):655–671, 2015.

[11] Andrei A Rusu, Mel Vecerik, Thomas Rothörl, Nicolas Heess, Razvan Pascanu, and Raia Hadsell. Sim-to-real robot learning from pixels with progressive nets. *arXiv preprint arXiv:1610.04286*, 2016.

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