POLITECNICO
MILANO 1863

SCUOLA DI INGEGNERIA INDUSTRIALE
E DELL'INFORMAZIONE

Executive Summary of the Thesis

Pose Estimation and Semantic Meaning Extraction for Robotics Using Neural Networks

Laurea Magistrale in Automation and Control Engineering - Ingegneria dell'Automazione

Author: Davide Figundio

Advisor: Prof. Andrea Maria Zanchettin

Co-advisors: Ing. Niccolo' Lucci, Prof. Paolo Rocco

Academic year: 2022-2023

1. Introduction

Perception is a fundamental task in robotics, that provides a control algorithm with knowledge and information on its surroundings through sensors and data processing. While RGB color cameras are one of the most readily available sensors, they are rarely used for this task. This is primarily because their outputs do not not directly reflect any physical quantity, requiring a difficult analysis process on a large amount of data.

Neural networks, and Convolutional Neural Networks in particular, are perfectly suited to solve these issues, due to their ability to parallelize and model complex and unknown functions through training. Because of this, the introduction of CNNs to tasks such as object detection and pose estimation has led to rapid improvements in accuracy and reliability, subsequently leading to an interest in applying these methods to perception tasks in robotics.

However, machine learning approaches bring a series of challenges, primarily due to their hard requirement for large amounts of training data, and the opaqueness of the resulting models to conventional analysis and understanding. Furthermore, the raw outputs of a pose estimation network must often be refined into more abstract information in order to be utilisied by the high-level planning algorithms commonly implemented in robotics applications.

In this thesis, we explore some possible solutions to these issues, by utilising a state-of-the-art 6D pose estimation neural network, EfficientPose[1], and implementing it in a robotics application. In particular, we begin by developing a method for efficiently generating training datasets starting from photographed backgrounds, using an "Augmented Reality" approach (section 2). Then we devise an approach for abstracting the semantic state of a scene exploiting the outputs of the trained network (section 3). Finally, we test the overall performance of our methods in real-world conditions, using our pose and state predictions to plan the movement of a robotic manipulator in performing a simple assembly task (section 4).

2. Augmented Reality Datasets

The main limitation of using synthetic datasets for training is the difficulty of reproducing real-world conditions. Since a simulated sensor and

simulated environment cannot re-create noise and unmodelled physical effects the same way a real sensor would, a model trained purely on sythetic data is not guaranteed to behave correctly in a real-wold scenario. This issue is commonly called "reality gap". Our approach for solving this problem is to make the training images as similar as possible to the real environment, while introducing a sufficient degree of variability in order to make the model adaptable to various different conditions, according to the principles of Domain Randomization[5]. The philosophy behind this approach is that most industrial and collaborative robotics applications are stationary, thus we can specialise the dataset, and therefore the network, on one particular environment.

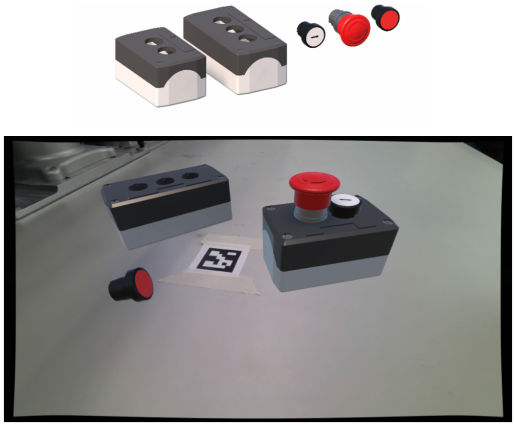

Figure 1: 3D models used for generating one of our AR datasets, and a sample image from the resulting dataset, that we called ButtonPose-near.

We therefore start from a set of photographs, captured from several different positions that are similar to those assumed in the testing scenario. In each photograph, the surface where the objects would be placed is highlighted using an ArUco[3] marker, making its pose easily detectable. We then generate the dataset as a set of images that use these photographs as backgrounds. For each image, we render the 3D models of each object on top of the backgrounds, setting their pose with a sequence of three roto-translations:

1. An initial roto-translation (t_s, R_s) from the camera reference to the position of the surface, given the ArUco marker.
2. A second, randomised roto-translation (t_r, R_r), dependant on the object's degrees of freedom. In our case, for objects placed on a surface, this is of the form:

$$t_r = \begin{bmatrix} x_r \\ y_r \\ 0 \end{bmatrix}, \quad \mathrm{R}_r = \begin{bmatrix} \cos\theta_r & -\sin\theta_r & 0 \\ \sin\theta_r & cos\theta_r & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

   where x_r, y_r, θ_r are extracted from uniform probability distributions.
3. A final correction transformation (t_c, R_c) that aligns the object with the surface, according to its geometry.

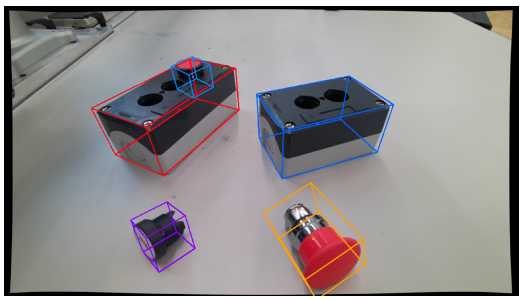

Figure 2: Example detections and pose estimations performed by an EfficientPose model trained on the ButtonPose-near dataset.

Object	AP	AD-S [mm]	ADD-S
2-slot	0.9994	2.6850	99.94%
3-slot	1.0	2.8913	99.95%
red button	0.9663	1.3550	95.84%
arrow button	0.9729	1.4384	96.47%
safety button	1.0	1.7229	99.84%
unknown button	0.9948	1.3384	98.74%
Average	0.9889	1.9052	98.46%

Table 1: Performance of EfficientPose trained on ButtonPose-near, measured using the Average Precision, Average Symmetric Distance (AD-S), and ADD-S[6] metrics.

We then use data augmentation methods that rescale and resize each image during training, and change its colors and grain. These methods further randomise the dataset, increasing its adaptability to new poses and light conditions[1].
Thus we generated several datasets using this method, allowing us to train and test the model's performance in different conditions and for various objects. In table 1 we show the performance of one of these, ButtonPose-near, and a sample real-world inference for the same dataset is shown in figure 2.

This variability in conditions gave us the opportunity to make a couple of observations. We noticed that the network's accuracy and reliability is vastly dependant on an object's size and distance from the camera. Our best results came from datasets where the objects were placed as close as possible to the camera while limiting occlusions.
Secondly, we noticed that our models, being trained on a small number of objects, are very susceptible to false positives due to the introduction of untrained objects into the image. This behaviour can be reduced to a certain extent by introducing "decoy" items into the generation procedure without labelling them in the dataset. In this manner, eventual false positives resulting from their appearance are recognized as such during training and corrected. While in theory this behaviour could be generalised by introducing enough variety of decoys into generation, the nature of black-box models makes this unpredictable, meriting further research.

3. Semantic Meaning Extraction

For many tasks it is necessary to abstract low-level information into high-level descriptions. The nature of these descriptions is specific to the task at hand, but a common situation is the assembly of a workpiece from its components. By tracking the each component's pose, we would obtain the state of the overall assembly. For this thesis, the components we considered are the set of modular buttons and boards previously depicted in figure 1. This procedure is generalisable to any set of components, as long as they remain identifiable throughout the entire task.
A high-level description for our application requires determining which buttons are in which slots. We handle this using a *threshold comparison* approach, which can be divided three steps:

1. Using the poses inferenced by the neural network, we compute for each button and each slot a *distance metric* that represents the button's "distance" from that slot.
2. We compare this metric with a *distance threshold*: if it is less than this value, the button is a viable candidate for the slot.
3. We resolve conflicts between multiple buttons assigned to the same slot and vice-versa.

We considered two possible *distance metrics*: Center-to-Center (C2C) and Average Symmetric Distance (AD-S). Both require two poses: the button's pose as estimated by the network, $(\hat{t}, \hat{R})$, and the pose the button would have if it was in the slot, (t, R). This second pose is computed using the network's estimate of the board pose that the slot belongs to. C2C is then the distance between the center of the two poses, $||\hat{t} - t||_2$, while AD-S is expressed as:

$$\text{AD-S} = \frac{1}{n} \sum_{x_1 \in M} \min_{x_2 \in M} ||(\mathrm{R}x_2 + \mathrm{t}) - (\mathrm{R}'x_1 + \mathrm{t}')||_2$$

where M is a set of 3D points belonging to the object, and n is the number of points considered. While C2C is much more computationally simple, it does not percieve errors in rotation, while AD-S is much more precise.

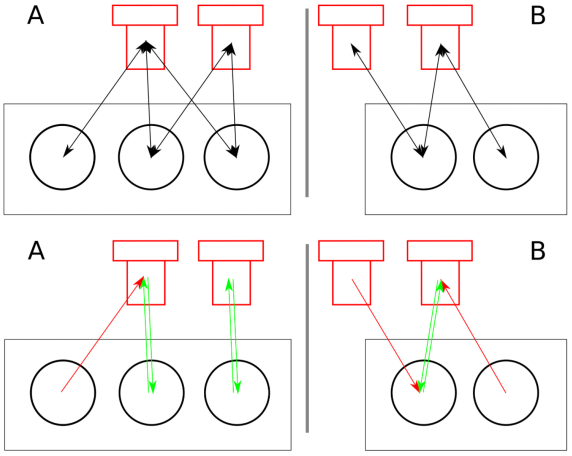

Figure 3: Conflicts that may appear when assigning buttons to slots for higher values of the threshold, and resolution using our method. Arrows represent possible assignments, green arrows confirmed assignments, and red arrows ignored assignments.

Our conflict resolution strategy becomes necessary for higher values of the *distance threshold*, when we may have multiple buttons assigned to the same slot, and vice versa. We solve this using an intuitive double-check approach:

1. For each button, we assign it to the closest slot within the threshold, if there is one.
2. For each slot, we assign it to the closest button within the threshold, if there is one.
3. For each assignment, it is confirmed only if it is reciprocated, and otherwise it is ignored.

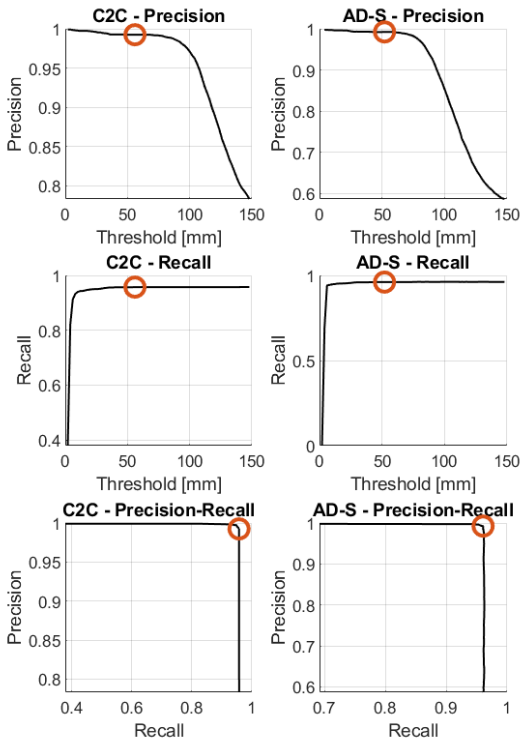

Figure 4: Precision, Recall and Precision-Recall for the C2C and AD-S metrics on the ButtonPose-near dataset. The red circle indicates the point corresponding to the optimal F1 score.

Metric	F1	Threshold
AD-S	0.9763	52 mm
C2C	0.9746	56 mm

Table 2: Optimal F1 scores and thresholds for the ButtonPose-near dataset.

This method gave us optimal results with minimal added complexity.
To evaluate our method, we considered a set of possible *distance thresholds* ranging from 2 mm to 15 cm and ran our algorithm on the testing datasets, since they already provided reliable ground truths. By then comparing our method's outputs to these ground truths, we obtained a confusion matrix for each threshold, and used these to compute a precision-recall graph. An example graph for the ButtonPose-near dataset is visible in figure 4.
The selection of the optimal *distance threshold* was performed by considering the F1 score, a balanced function of precision and recall described by:

$$\mathrm{F1} = 2 \times \frac{\mathrm{Precision} \times \mathrm{Recall}}{\mathrm{Precision} + \mathrm{Recall}}$$

We considered the optimal value to be the one with the highest F1 score. Example values for the ButtonPose-near dataset are visible in table 2. As can be seen, we obtained high values of the threshold, above 5 cm in all testing conditions. The reason for this behaviour is probably because our dataset represents an ideal condition, where a hypothetical manipulator has not commited any mistakes in picking up buttons and inserting them into the slots. If failed attempts were considered in the dataset, the optimal threshold would probably be lower, and the AD-S method, being more sensible to errors in rotation, would obtain a greater advantage in precision compared to C2C.

4. Real-World Testing

Testing was performed using a Doosan A0509s robotic manipulator equipped with a pneumatic gripper and an Azure Kinect camera. To drive the robot, we implemented a system that builds behaviour trees[2] based on previous demonstration of actions and their effect on the scene, as visualized through the camera.
The perception phase therefore consists of three tasks:

1. From the RGB image provided by the camera, we use the network to detect objects and their poses.
2. We apply our semantics extraction method to understand the interactions between the detected objects.
3. We use these interactions to build a list of predicates that describe the scene.

These predicates are first-order logic functions that can be either true or false, and are updated each time the system recieves new information. Those that result as being true describe the scene's *state*.

Predicate	is true when..
IsGripperEmpty(gripper)	no objects in the gripper
IsGrasped(button, gripper)	button is grasped by gripper
IsButtonInSlot(button, slot)	button is inserted in slot
IsSlotEmpty(slot)	no buttons are in slot

Table 3: List of predicates used to describe the state for our application.

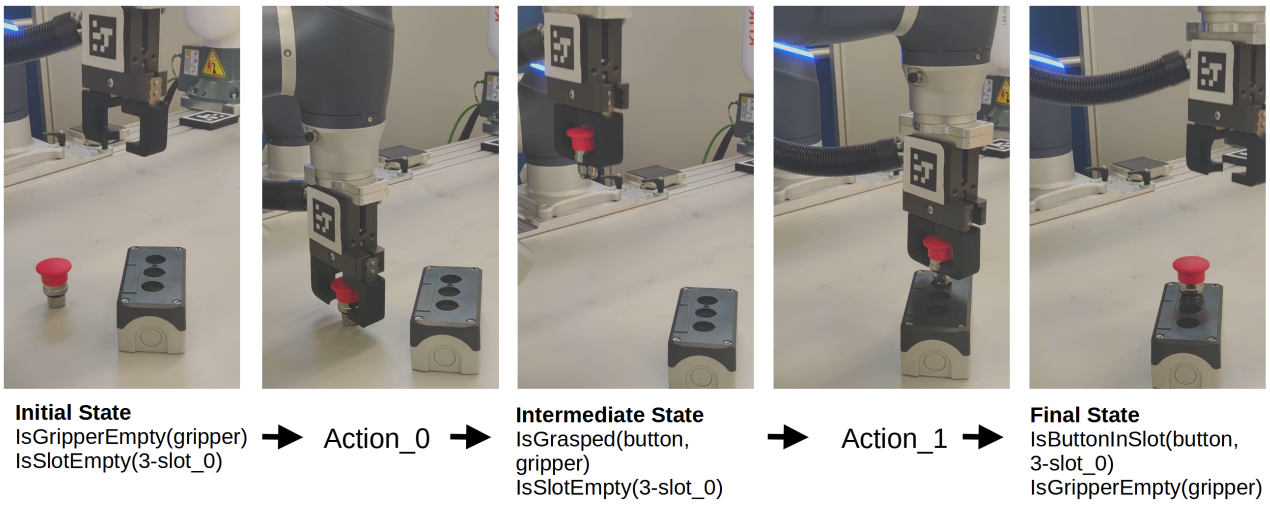

Figure 5: Evolution of the state and actions for the example task of picking up a button and inserting it into a slot.

The experiments are then divided into two main phases: teaching and evaluation. In the teaching phase, we show the robot how to perform actions through kinesthetic demonstrations. These demonstrations are referenced with respect to a manually set objec. Consequently, the same action can be repeated even if the object is in a different position: for example, when picking up a button, we set the reference landmark to the button itself. To evaluate the *state* before and after the demonstration, we use PDDL[4] to build a set of preconditions and effects expressed in predicate form, which combined with the low-level robotic actions form a *skill*. For the example in figure 5, we would obtain two *skills* as defined below:

```
(:action action_0
    :parameters(
        ?Gripper1 - Gripper
        ?SmallButton1 - SmallButton
    );end_of_parameter
    :precondition (and
        (is_gripper_empty ?Gripper1)
    );end_of_precondition
    :effect (and
        (is_grasped ?SmallButton1 ?Gripper1)
        (not (is_gripper_empty ?Gripper1))
        (increase (total-cost) 50)
    );end_of_effect
);end_of_action

(:action action_1
    :parameters(
        ?SmallButton1 - SmallButton
        ?Gripper1 - Gripper
        ?Slot1 - Slot
    );end_of_parameter
    :precondition (and
        (is_grasped ?SmallButton1 ?Gripper1)
        (is_slot_empty ?Slot1)
    );end_of_precondition
    :effect (and
        (is_gripper_empty ?Gripper1)
        (is_button_in_slot ?SmallButton1 ?Slot1)
        (not (is_grasped ?SmallButton1 ?Gripper1))
        (not (is_slot_empty ?Slot1))
        (increase (total-cost) 50)
    );end_of_effect
);end_of_action
```

The union of all actions, objects and predicates is called the *domain*. In the evaluation phase we instead build the *problem*, which combines a representation of the initial state and the goal state we would like to achieve. By providing a PDDL planner with a *domain* and a *problem*, we then have it compute the *plan*: an ordered sequence of actions that brings us from the inital state to the goal state. We then build a behaviour tree by combining the individual trees for each action in the order defined by the planner, and drive the robot by evaluating the tree. We had the robot perform a simple assembly task, where it had to pick up a button in various different positions and insert it into a chosen slot on a board. The results of this experiment are presented in table 4. The main issue encountered in testing was the prevalence of small rotation errors in the neighborhood of $\pm 5°$, which could turn into awkward pick-ups and consequently failed insertions. These were much more present with smaller objects, leading us to believe that the performance of the overall system is highly dependant on that of the underlying network.

Button Type	Small	Large
Total Attempts	10	10
Successful Pick-Ups	7	10
Successful Insertions	5	7

Table 4: Experimental results for our test assembly task.

5. Conclusions

Overall we believe that machine learning is a very powerful tool for robotics application, and that trained neural networks can achieve remarkable performance in perception tasks. However, we do not believe that they are sufficient for tasks that require high precision and reliability, or tasks that involve small objects, or objects that are difficult to identify in other ways.
Furthermore, it is difficult to overcome their black-box nature, meaning that their performance may be compromised when applied in different working conditions with different objects. This is particularly noticeable with our approach, which when faced with new working conditions would very likely require a time consuming re-training process to allow the network to adapt. This specificity on one environemnt is therefore the main disadvantage of our approach, but also what makes its high performance possible.
Future improvements on our work could include:

- Comparing the performance of a network trained on one of our generated datasets with a network trained on real-world data.
- Verifying the possibility of avoiding false positives by training a model to ignore a wider variety of objects.
- Verifying whether including multiple different environments in the background images during dataset generation improves generalisation to new environments.
- If future pose estimation approaches increase performance in a significant manner, verifying whether they obtain the accuracy required for high precision robotics applications.

References

[1] Yannick Bukschat and Marcus Vetter. Efficientpose: An efficient, accurate and scalable end-to-end 6d multi object pose estimation approach, 2020.

[2] Michele Colledanchise and Petter Ögren. *Behavior Trees in Robotics and AI*. CRC Press, jul 2018.

[3] S. Garrido-Jurado, R. Muñoz-Salinas, F.J. Madrid-Cuevas, and M.J. Marín-Jiménez. Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, 47(6):2280–2292, 2014.

[4] M. Ghallab, A. Howe, C. Knoblock, D. Mcdermott, A. Ram, M. Veloso, D. Weld, and D. Wilkins. PDDL—The Planning Domain Definition Language, 1998.

[5] Josh Tobin, Rachel Fong, Alex Ray, Jonas Schneider, Wojciech Zaremba, and Pieter Abbeel. Domain randomization for transferring deep neural networks from simulation to the real world, 2017.

[6] Yu Xiang, Tanner Schmidt, Venkatraman Narayanan, and Dieter Fox. Posecnn: A convolutional neural network for 6d object pose estimation in cluttered scenes, 2017.