Text Based Pick and Place

Adrián Aparisi Seguí, Guglielmo Borzone, and Nathan Savard *

Submitted: 31 May 2023

Abstract

This paper presents a text-based approach to perform pick and place tasks. The proposed method combines language processing, computer vision and robot control techniques in order to understand prompted texts, find objects, and control robot movements. With this proposed pipeline, the aim is to enable a user with no robotics knowledge to make the robot move to perform pick and place tasks.

Text-based, YOLO, CLIP, pose estimation, pick and place, RRT.

1 Introduction

Modern robot interfaces focus on using code or psuedo-code based implementations of routines and actions. These setups require a knowledge of the control software and how to use it, as well as an understanding of common coding practices. In this project, we propose an alternative control method which uses text-based inputs to control a robot for pick and place tasks.

In this project, we focus on a specific use case to create a proof-of-concept system. Here, we isolate the scenario with a collaborative robot, where the robot and human operator have a visual view of the workspace. We also assume the workspace will contain a set of tools, such as hammers, screwdrivers, and pliers, which should be moved to a desired location. This simple pick-and-place scenario is used to make reasonable assumptions for the development of the system within our time constraint.

1.1 Background

Robots are used increasingly to automate complex tasks in the workforce. Companies have developed many ways to control these robots to set up tasks in the workspace. Additionally, there is a shift towards collaborative robots that can work alongside humans to accomplish tasks that could be impossible without a combined effort.

Currently, humans interact with robots through a handful of interfaces to direct robots in the workspace. Robots can be accessed through the teach pendant, which gives common commands to be connected as blocks to create a program. This requires a set of pre-programmed commands and positions that can be called in order to accomplish the task. Alternatively, learning by demonstration is another method which can be applied. This is where an operator can manually guide the robot through a set of motions to be repeated. For more complex tasks, a robot can be controlled directly through programmable interfaces, where engineers can directly control robot parameters and code commands.

These methods require a working knowledge of the robot and its capabilities. The simpler methods are powerful tools for quick setup, but are not capable of performing some complex or more flexible tasks. It can take a long time to develop programs for more complex problems, requiring a more in-depth knowledge of robot control theory.

This paper investigates another, more intuitive approach to robot control combining computer vision and AI techniques with natural language. This could provide an instinctive interface between robots and humans, allowing humans to control robots without the reliance on inflexible control schemes or complex coding interfaces.

1.2 Literature review

Multiple advances have been made during the last years regarding artificial intelligence and computer vision. Thus, classical methods in areas such as object detection or image classification have been replaced by AI approaches.

In order to find objects in a scene there are several methods or algorithms that are currently widely used. For example, the YOLO (You Only Look Once) algorithm, since its first release in 2015 by Redmon et al[1]. has shown great performance in object detection. Since then, there have been multiple new upgrades of this algorithm until its eighth version YOLOv8.[2]

Pix2Seq is another example that acts as a simple and generic framework for object detection based on neural networks and deep learning. The framework is based on converting the object detection task into a language modeling task based on the input pixels. Although the results for this network are promising, the authors state that it is not fast enough for real-time applications. It is also a general model that does not use previous knowledge about the input. This is something that we want to use for our experiment, since we

*All authors contributed equally

know the environment and the objects that are being classified.[3] For these reasons we ruled this model out in favour of YOLO.

1.3 Project Goal

We believe it is possible to combine these new algorithms and models to create an intuitive and flexible robot control interface with humans. The goal of this project is to use natural language as the input for a pick-and-place task. In this application, we want to allow operators to command a robot to move or organize objects in a shared workspace, without the need for more complex user input.

Our pipeline follows the following procedure:

- First, we break up the prompt sentence into its key ideas using Text Parsing (section 2.1).
- Second, we find candidate objects in the camera using Object Detection (section 2.2).
- Third, we compare those candidate objects to the key ideas from the text prompt using Text Matching (section 2.3).
- Fourth, we calculate the pose of the selected object (section 2.4).
- Finally, we pass the desired pick and place poses to the robot controller to perform path planning and execution (section 2.5).

Each step in this pipeline prepares the data for the next step, allowing information to be passed between stages to create the final output.

2 Methods

In this section, we discuss the methods we applied to achieve the desired goal.

2.1 Text Parsing

The first step in our process is text parsing: to break down the prompt sentence into its key ideas which can be applied to the pick and place task. To accomplish this, we implemented the python library spaCy[1], which breaks down a given text string into key linguistic features. These features include its "tags" describing how the word acts in the sentence and "parents and children" to describe their relationship with other words. More information about how we used these tags can be found in Appendix C. We use this information to construct a tree of words, as seen in Figure 1. This tree shows which words belong to others. From there, we focus on a few specific tags of importance:

- Direct Object (dobj): Specifies the object being directly addressed by the verb. We use this as our "pick" item.

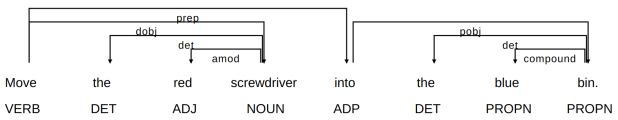


Figure 1: Linguistic Features of a Sample Sentence

- Preposition Object (pobj): Specifies the object being addressed by the preposition of the verb. We use this as our "place" location.
- Adjectives (amod): add additional information about an object, such as color. We pass these alongside their parent object to distinguish between similar objects in the scene.
- Compound (compound): used similar to adjectives, but they address compound words that are used together to describe an object.

By parsing the sentence in this way, we find the objects of interest, such as "screwdriver" or "pliers" and their descriptors such as "red", "large", or "metal", we can pass this information to the next steps in the pipeline. The pick object is passed directly to the object detection algorithm to find objects matching the description. The descriptors can be sent to the text matching to select between candidate objects.

2.2 Object Detection

This section describes the steps we followed to perform object detection using the You Only Look Once (YOLO) object detection model [4].

2.2.1 Data gathering

To find specific objects, tools in this case, a dataset of 320 images was compiled, containing objects of six different classes: pliers, hammers, tape measures, screws, screwdrivers and wrenches. The dataset includes images with annotated bounding boxes for each object of interest, these bounding boxes have been generated with the open-source software labelImg [2] as described in Appendix A. The images have been divided in two groups, training and evaluation for the YOLO algorithm [5].

2.2.2 YOLO Implementation

The widely used YOLO [6] algorithm was implemented in order to place bounding boxes around objects in an image. After training the model, YOLO executes a single convolutional network and simultaneously predicts the corresponding bounding boxes and the class probabilities for those boxes.

This will allow the pipeline to isolate just the objects of interest matching the class name with the input prompted text.

[1] https://spacy.io/

[2] https://github.com/heartexlabs/labelImg

2.2.3 Training

YOLOv8 [3] comes with five model variations for object detection labelled by the characters *n, s, m, l, x*, with *n* being the fastest and smallest and *x* being the largest, most accurate, but slowest variation. For this task, we use the *l* model for its balance of speed and performance.

There are pre-trained models for these options, as described in Appendix B, that allows the YOLO model to find generic objects. However, for our application, as mentioned, we use self-defined classes that are not present on the models that YOLO comes pre-trained with.

2.3 Text Matching

To further distinguish between similar objects, we apply the text-image matching "Contrastive Language-Image Pre-training" (CLIP) model. This model was developed by Open-AI as a way to learn image classification using natural language as a supervisor[7]. The general approach is to simultaneously train a text encoder alongside an image encoder, using the similarity between the produced vectors to train each model in turn.

For our implementation, we used a pre-trained model provided by Open-AI to perform our text-image matching. In this model, they used a transformer architecture to train the text encoder, and a ResNet image encoder[7].

This combined model takes inputs of an image and a text string, outputting a similarity score between the two. By running the algorithm on multiple images, we can take the soft-max of the outputs to pick the best matching combination.

We use the CLIP as a image-caption model by forming a sentence from the key information[7]. This can be formatted into a template sentence such as "A centered photo of [Object of Interest]," where we fill in the information retreived from our text-parsing operation. Because our object detector, YOLO, creates bounding boxes centered on the item of interest. The keyword "centered" will help eliminate interference from other objects. We insert the descriptors and object into this template to match the sentence with an image. For an example, if the query object is a "screwdriver" with the descriptors "small" and "red", then the query sentence would be "A centered photo of a small red screwdriver."

2.4 Pose Estimation

Once we successfully found the object of interest in the image, we proceed with pose estimation to find the center of the object in 3D space and its orientation so that we can grasp it with the robot.

For this task, we start from two assumptions:

- The object of interest will be the biggest visible blob in the bounding box.
- There is no overlay or underlay with any other object.

To find the orientation and the location of the blob we will use principal component analysis (PCA) [**?**]:

First we pre-process the image: we remove the background, using static background subtraction. We then threshold the image and convert it to binary.

Principal component analysis is a method widely used for dimension reduction, that computes the mean, the eigenvectors, and the eigenvalues of a matrix. Using this algorithm on our pre-processed image will return us these values. In this case, the mean represents the location of the center in the frame, and the eigenvectors and eigenvalues represent the orientation and shape of the object as shown in Figure 2. We perform this in the bounding box found using YOLO to find the local pose of the object.

Once this data is obtained, the location of the bounding box in the original image has to be added to the local center to translate it to the position it has in the original image.

Since the camera is calibrated with a known distance to the plane, we can use the pinhole model to compute the *X, Y, Z* position in camera frame.

We then use the transformation matrix T^{TCP}_{camera} to find the same point in the end effector frame of the robot system, we finally use the angle of orientation to compute the rotation matrix that will align the end effector to the object we are interested in.

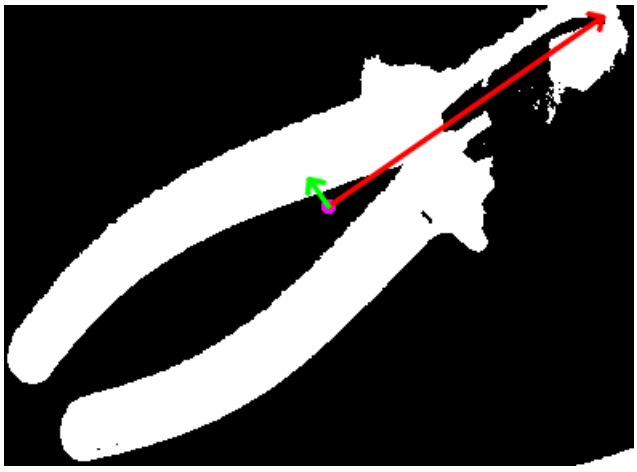


Figure 2: PCA output

2.5 Pick and Place

Once we have the global pose of the object, we can plan a robot trajectory to pick it up and move it to a desired location. In our experiment, we broke down our task in moving the robot to the previously computed pick pose, descending until contact with our plane, lifting it a set distance, and then moving either to a custom place position from the prompt or to a pre-defined pose. RRT and collision detection path planning functions

[3] https://github.com/ultralytics/ultralytics#known-issues–todos

from RobWork are used to find the grasp positions and valid path from one position to the next.

3 Results

3.1 YOLO training

As said in section 2.2.1, to train our YOLO model more than 300 images have been gathered and divided in two groups: 286 training images and 32 evaluation images.

Results from this training can be seen in figure **??**, where *box_loss* refers to the loss in bounding box generation and *cls_loss* refers to the loss component associated with the classification of object classes.

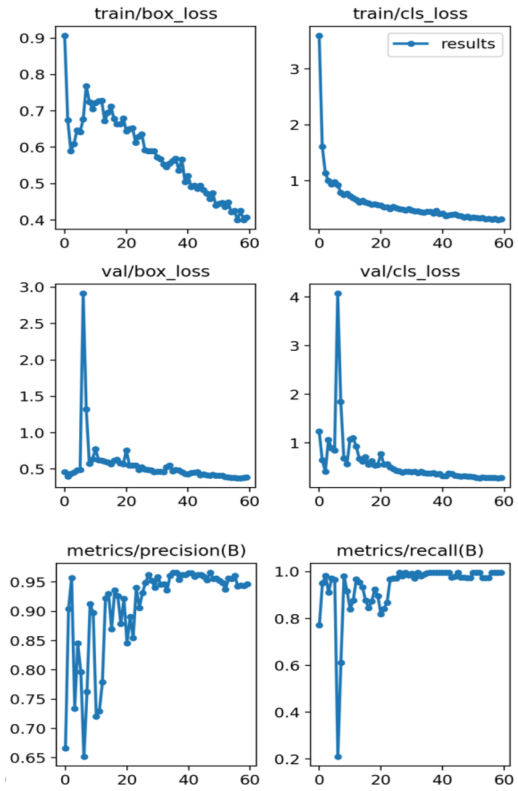


Figure 3: Results obtained from training the model

3.2 YOLO performance

To test YOLO in our pipeline, we took a set of 90 additional images to analyze the number of objects the algorithm is able to successfully find.

We realized that the performance of this section of the pipeline was highly influenced by the the size of the image. We found that taking images from a wider perspective reduced the performance. Therefore, we cropped the image to a fixed size to focus only in the area of interest where the tools actually were.

For example, in Figure 4 it is shown how, without cropping the image, the algorithm misclassifies the screwdriver labelling them as pliers instead.

The results of running YOLO on this set of images can be seen in table 1.

Without cropping images				
No. of images	Elements to be identified	Correct matches	No identified	wrong classification
90	394	243	151	94
Cropping images				
No. of images	Elements to be identified	Correct matches	No identified	wrong classification
90	394	325	64	60

Table 1: YOLO Results for cropped and non-cropped images.

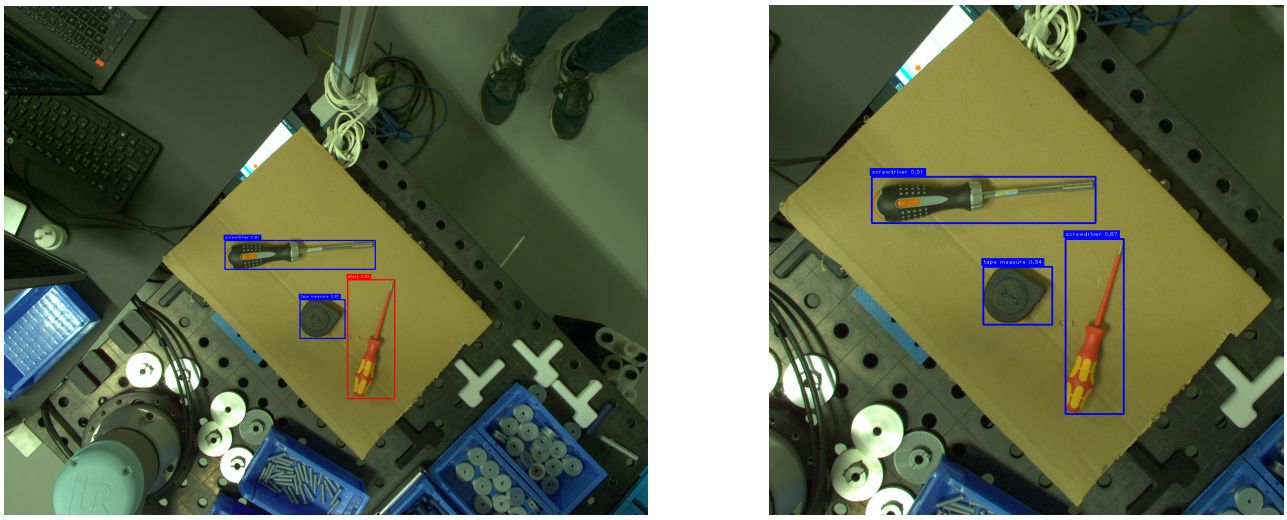
3.3 CLIP Results

Because we used a pre-trained CLIP model for our implementation, we ran tests to characterize the effectiveness of the model in this specific scenario. To test this, the CLIP model was run on a set of images, each containing a different variation of one object class, as seen in Figure 5. Each object in the image was then distinguished with three attributes:

- Color: the major color(s) of the object;
  such as "red" or "blue"
- Material: What the object is made out of;
  such as "wood" or "metal"
- Type: The specific type of the object;
  such as an "allen wrench" or "adjustable wrench"

For example, the leftmost wrench shown in Figure 5 would be classified as a "grey metal combination wrench". A full list of the data set and attribute labels can be found in Appendix D. These three attributes were then split up into individual prompts for the following tests.

Each subsection of the image (shown in Figure 8) was then passed to the CLIP model along with a prompt taken to identify one specific object. Additionally, each attribute was tested in combination with another, to test for the affects of more complex query sentences. For example, a combination of color and material could be the "brown wood hammer". The results of this test can be seen in Table 2. In this table, the diagonal cells show the accuracy of a single attribute, while the upper triangle shows the accuracy of combined attributes. These results show that color is the best single attribute to correctly identify the object, with an overall accuracy of 66%, followed by material with an accuracy of 55%, and then by type, which only had a 28% overall accuracy. This lead to an overall accuracy of 45% across all trials. Interestingly, combining attributes did not have a consistent effect on the results. As seen in Table 2, The accuracy of a color combined with a material attribute lowered the overall accuracy to 50%, less than either in isolation. While



(a) Non-cropped image.

(b) Cropped image.

Figure 4: Object detection output with cropped and non-cropped images.

Figure 5: An image of multiple wrench variations used to validate the CLIP model

other combinations brought the overall accuracy to a point between the two individual attributes.

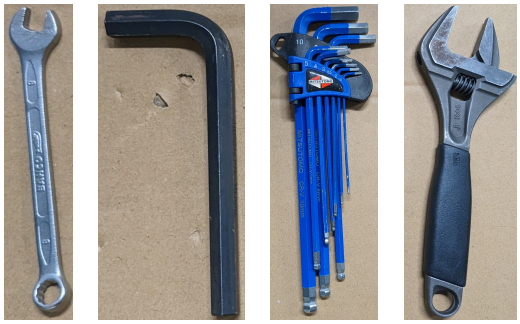


Figure 6: Objects split into image subsections

We also separated the results into the different object classes being identified, as shown in Table 3. These results show that some objects are easier to distinguish between for the CLIP model. The best performing object class was the hammer, with an overall accuracy of 78%, followed by the wrenches at 54%. The worst performing classes were the measures, pliers, and screwdrivers, all below 40% accuracy. These results could be because of the types of variations within these objects. The hammer set varied in color, material, and type, while the other classes shared many attributes (such as all wrenches being "metal").

-	color	material	type
color	66%	50%	55%
material	-	55%	28%
type	-	-	28%

Table 2: CLIP accuracy for different attributes

Class	Accuracy
Hammer	78%
Measure	33%
Pliers	38%
Screwdriver	38%
Wrench	54%

Table 3: Accuracy to distinguish each object class

3.3.1 Expanded CLIP Tests

To further explore the use of CLIP in this application, we tested it in extreme cases where it may be used. One test we performed was to run CLIP on unlabeled objects, rather than through labelled objects found by the object detection model. This meant we would test our query text on every object in an image, not just those matching the sample object. We ran the same accuracy test on the data set from Section 3.3.

The results of this test can be seen in Table 4. These results show that the overall accuracy decreases as a result of the change by an average of 61%. However, CLIP is still able to successfully distinguish many objects, even without the added object label.

-	color	material	type
color	44%	28%	33%
material	-	28%	17%
type	-	-	17%

Table 4: CLIP accuracy for different attributes without object label

Splitting these results by the input object, we see that some objects were affected more heavily than others, as seen in Table 5. These results show that the accuracy for some classes, like the hammer or wrench were heavily affected. The hammer identification was reduced to 0%. However, some classes were unaffected, such as the pliers and the measures.

Class	Accuracy
Hammer	0%
Measure	33%
Pliers	38%
Screwdriver	33%
Wrench	29%

Table 5: Accuracy to distinguish each object class

These tests reveal the importance of the object detection step used in the full pipeline. While some objects can be distinguished with CLIP alone, some (such as hammers) fail completely. With the addition of the object detection step, we greatly increase the overall accuracy.

3.4 Overall Pipeline Success

The results of the overall pipeline were not as good as the results from running the single steps alone. The two main steps that diminished when moving into the lab were: YOLO classification and robot grasp planning.

When we tested YOLO to verify its consistency and how well it would perform, we preprocessed the image to crop the image and remove excess information. In the lab experiments we ran the pipeline without this preprosessing step to avoid added complexity. It resulted in a lower score, probably as a result of the added noise on the image and the reduced zoom.

We kept the robotic grasping part of our project as simple as possible so that we could focus on the vision part in more in depth, as a result the robot grasping was not as refined as it could be. The setup that we use had a camera attached, searching for a collision free path would often fail for this reason.

4 Discussion

In this section, we discuss the results of the experiments and the implications on the impact of this work.

4.1 Pipeline Discussion

In this project we have developed a pipeline to find objects from a prompt and move them to a desired location. This process was broken into individual steps, each of which implemented a unique model. As each section has individual errors, those errors compounded when combined together. We found that the outcome worked as a proof of concept, but was not a fully functional utility.

We were unable to extensively test the pipeline due to time constraints, but we were able to interact with the system experimentally. During these experiments, we found some success in its ability to understand commands and perform a pick and place task. The most common errors were object misclassification and failed grasp/path planning.

4.2 Environment and Setup Limitations

When developing this project, we made some assumptions on the setup or environment. This lead to limitations in its flexibility.

Because our object detection model had a limited training data set, it resulted in a model that was specialized to this specific task, but could not be generalized broadly to other tools.

Furthermore, this dataset has been created with certain light and environment conditions. Using the same model in another workspace can lead to different results. Testing how these results can vary according to these changes in the environment is not in the scope of this paper.

Ambiguities on the input text should be considered. This pipeline has been tested according to sentences that the authors had introduced. Therefore, using others sentence structures could lead to unexpected errors.

This project requires an image of the background to find the contours of the objects. This is used for static background subtraction when performing pose estimation. Therefore, using another setup would require another initial background configuration.

Our robot environment was also constrained to a predefined workspace, where few modification were allowed. This lead to unsuccessful path planning to pick position due to the large camera mounted on the robot, Also, narrow gripper aperture that did not allow us to correctly pick certain wide objects such as pliers or tape measure.

4.3 Model Limitations

We found that each model was limited in its capacity. It is important to consider how each model can process information to understand the consequences on the overall system.

Our text-parsing algorithm from *spaCy* was limited by the complexity of the sentences. The model would sometimes mislabel certain words such as "adjectives" being classified as "compounds", which altered how the information was passes to later steps. Our implementation also made assumptions on the sentence structure that could not be guaranteed from a different user. For example, the algorithm only finds the first preposition object, so compound sentences could not be parsed.

Object detection is inherently limited by a set number of classes. This means the model must be trained on a task-specific dataset to be used. To generalize the method, it would require either a much larger dataset or an entirely different model for the task.

The CLIP model was also constrained by the input text. There are some intuitive attributes that we could not test with CLIP, such as *position*. For example, CLIP cannot find the "screwdriver on the right" because it compares each item in isolation, not relative to each other. This also made it difficult to distinguish using *size*, although it could sometimes distinguish between them.

5 Future Work

: The model we built has high scalability because it is composed of different parts that are almost independent between each other. This leave us with many possibilities regarding future work. After the tests we run in the lab we realized some more components that could be developed to improve reliability and the general results of the project

- YOLO has been trained on 6 classes (hammer, pliers, wrench, measurement tape, bolt, screwdriver).This can be expanded to classify more types of objects.
- All the parts composing the pipeline can be upgraded or exchanged with more advanced or newer methods with minimal modification to the main part of the project.
- The text query could be in future swapped with a query obtained by voice, to make it simpler and faster for the operator.
- Improved image pre-processing could improve the overall score of the YOLO model when run in the lab. Specifically, zooming and cropping to remove extra information
- The grasp planning would often fail because it could not find a collision-free path to the specified pose. Adding a π rotation on the yaw angle to generate a second pose for the robot to try would improve the results.

6 Conclusions

Our pipeline demonstrates how existing tools can be combined to address a new use case to create an intuitive user-robot interface. Our method shows the possibility of robot control from simple text inputs rather than complex code-based implementations.

Our results show that our model is reasonably accurate within the scope given: tools in a controlled workspace. We were able to successfully pick and place many tools in the experimental setup, with some constraint on the initial setup (e.g we had to orient the object in a way that would ease the grasp planning).

Our method is limited in scope by each of the models used in the process. For example, the object detector is limited to a small vocabulary of items, meaning it is not immediately expandable to a broader application. With these deficiencies, the algorithm must be tuned for each possible use-case that it is applied for. This would include taking new training images and training the YOLO model in that particular workspace.

There are many ways to expand the model, such as exchanging particular steps in the pipeline for alternative models. This could include simple swaps, such as upgrading to a newer version of YOLO, or could change larger parts, such as picking a zero-shot capable detector. These future works could help to improve the system for more general use.

Acknowledgments

We would like to thank our advisors Anders Glent Buch, and Frederik Hagelskjær for their support on the project, and for allowing us to use the robotic setup for testing our experiments.

References

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, You only look once: Unified, real-time object detection (2016), arXiv:1506.02640.

[2] P. Jiang, D. Ergu, F. Liu, Y. Cai, and B. Ma, A review of yolo algorithm developments, *Procedia Computer Science* **199**, 1066 (2022).

[3] T. Chen, S. Saxena, L. Li, D. J. Fleet, and G. Hinton, Pix2seq: A language modeling framework for object detection, *arXiv preprint arXiv:2109.10852* (2021).

[4] Y. Amit, P. Felzenszwalb, and R. Girshick, Object detection, *Computer Vision: A Reference Guide* 1–9 (2020).

[5] C. Liu, Y. Tao, J. Liang, K. Li, and Y. Chen, in *2018 IEEE 4th Information Technology and Mechatronics Engineering Conference (ITOEC)* (2018), 799–803.

[6] J. Redmon and A. Farhadi, Yolov3: An incremental improvement, *arXiv* (2018).

[7] A. Radford, J. W. Kim, C. Hallacy, A. Ramesh, G. Goh, S. Agarwal, G. Sastry, A. Askell, P. Mishkin, J. Clark, et al., in *International conference on machine learning* (PMLR, 2021), 8748–8763.

A YOLO training

As said in section 2.2.1 we created our own dataset to train YOLO. LabelImg (see figure 7) is an open source software commonly used to generate datasets for AI applications. In this case, it has been used to generate our own dataset that we used to train YOLO in order to obtain a model able to find tools.

B Existing YOLO Models

The pre-trained models that YOLOv8 comes packed with are Object Detection checkpoints trained on the COCO detection dataset with an image resolution of 640p; instance segmentation checkpoints trained on the COCO segmentation dataset with an image resolution of 640 and image classification models pretrained on the ImageNet dataset, with an image resolutions of 640, 640 and 224 respectively (see table 6).

C Text Matching Implementation

Text matching was performed using the spaCy library, using their linguistic feature detection. For this, we used their pre-trained model. We specifically used the "Syntactic dependency" to describe words. These show the relationship between words of a sentence, rather than just the parts of speech.

Because of this, our pipeline requires statements formatted as commands with a primary verb, a direct object, and a preposition object or a single direct object to complete. We included here some example failed queries and wrote an explanation why they would fail in Table 7.

Query	Reason for failure
"Grab that blue screwdriver and then put it on top of the red screwdriver."	multiple prepositions
"Grab that blue screwdriver and then put it on the red one."	ambiguous object "one"

Table 7: Sample failed queries

D CLIP Test Data Set

This section shows the data input into the CLIP tests, as used in Section 3.3. Here, we used a set of images, each with a set of objects belonging to the same class. We then ran CLIP to see which objects could be distinguished. Objects in Table 8 are labelled starting with 1 with the leftmost object, increasing by 1 when selecting the next object to the right

E Code

All the code we used is on this github repo.

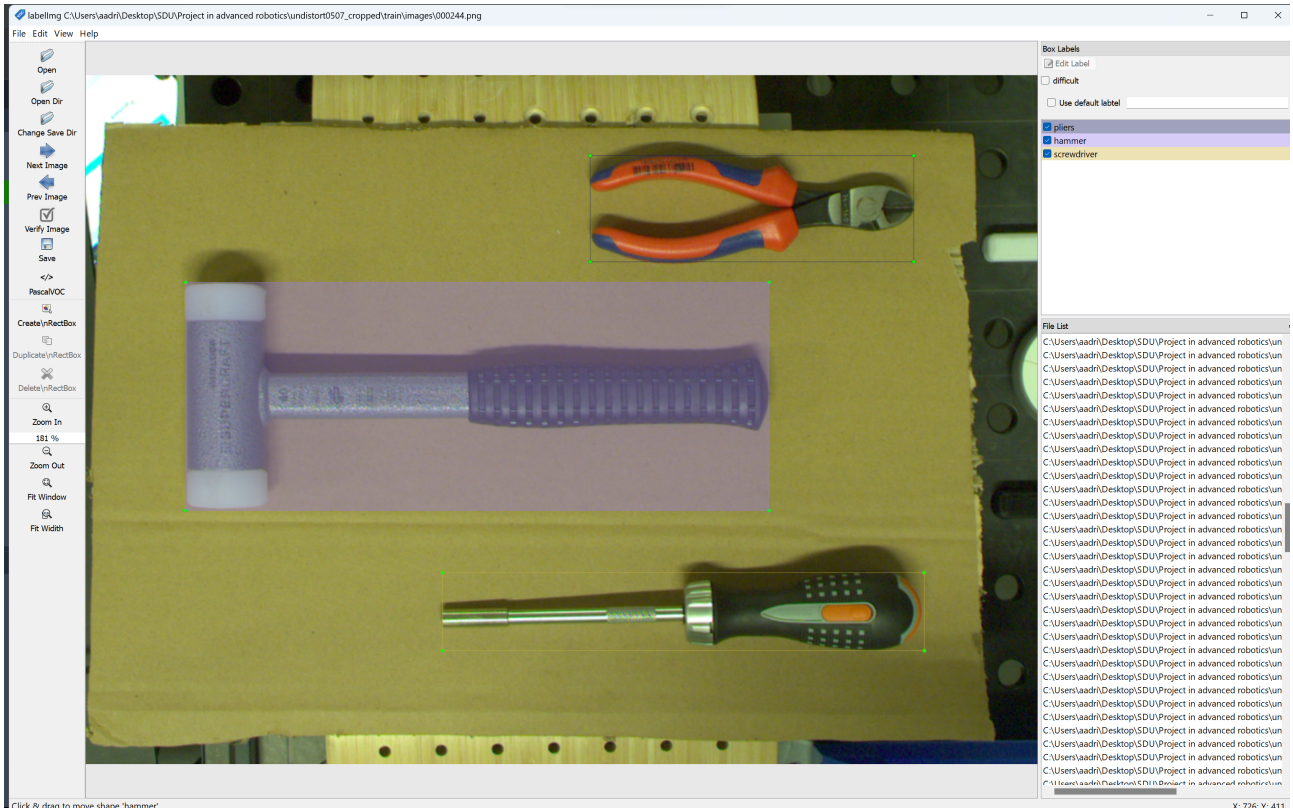


Figure 7: Caption

Model	size (pixels)	mAPval 50-95	Speed CPU ONNX (ms)	Speed A100 TensorRT (ms)	params (M)	FLOPs (B)
YOLOv8n	640	37.3	80.4	0.99	3.2	8.7
YOLOv8s	640	44.9	128.4	1.20	11.2	28.6
YOLOv8m	640	50.2	234.7	1.83	25.9	78.9
YOLOv8l	640	52.9	375.2	2.39	43.7	165.2
YOLOv8x	640	53.9	479.1	3.53	68.2	257.8

Table 6: YOLOv8 pretrained models.*

*Source: https://github.com/ultralytics/ultralytics

(a) Non-cropped image.

(b) Cropped image.

(c) Cropped image.

(d) Cropped image.

(e) Cropped image.

Figure 8: Full Images used for CLIP testing

Object	Color	Material	Type
Hammer 1	brown	wood	peen
Hammer 2	grey	metal	mallet
Hammer 3	orange	plastic	nail
Measure 1	white	wood	stick
Measure 2	black	plastic	tape
Measure 3	yellow	rubber	tape
Pliers 1	red	rubber	snap
Pliers 2	blue and red	plastic	cutter
Pliers 3	blue and red	plastic	needle
Pliers 4	black	plastic	cutter
Screwdriver 1	black and orange	plastic	female hex
Screwdriver 2	red	rubber	flat
Screwdriver 3	black	plastic	hex
Screwdriver 4	red and black	rubber	philips
Wrench 1	grey	metal	combination
Wrench 2	black	metal	allen
Wrench 3	blue	metal	allen
Wrench 4	black	plastic	adjustable

Table 8: Caption