Budapesti Műszaki és Gazdaságtudományi Egyetem

Villamosmérnöki és Informatikai Kar

MIT Tanszék

Márton L. Gellér

Recyclable Material Sorter Robotic Arm

Thesis

SUPERVISOR

Dr. Bence Márton Bolgár

BUDAPEST, 2023

Table of Contents

Hungarian Abstract (Kivonat)

A szakdolgozat egy robotkar fejlesztését mutatja be, amely képes az újrahasznosítható szemét szétválogatására. A dokumentáció ismerteti a robot mechanikai részei létrehozásának és az elektronikával való összeszerelésének folyamatát, továbbá tárgyalja egy objektumfelismerő modell mélytanulással történő betanításának menetét két különböző adatbázisból, amely lehetővé teszi, hogy egy Raspberry Pi zsebszámítógép kamerája felismerje a különböző típusú szemetet. Bemutatja annak a folyamatnak a programozási nehézségeit, amely során a kar egy monokuláris mélységbecslés alapján képes megfelelő távolságra elnyúlni a szemétért, és azt felvenni. Emellett a dolgozat feltárja az ehhez hasonló robotkarok sokoldalú alkalmazási lehetőségeit és az ezekhez társuló lehetséges, jövőbeli fejlesztéseket, miközben megvitatja a projekt motivációját.

Introduction

Nowadays, an issue most households and individuals face almost every day is selecting and sorting out recyclable rubbish correctly. Distinguishing non-recyclable garbage from recyclable trash can appear quite taxing sometimes. If recyclables are not sorted properly within the households, several further problems can arise. Let us be honest, we are humans, errors are made even unintentionally. For example, throwing a plastic cup with leftover coffee in it into the bin for recyclable plastic might cause further issues. A publication from 2018 [1] shows, that "*75% of the waste can be recycled, yet the national recycling rate was *only* 32.1%",* due to mistakes in sorting. The difficulties caused by not properly classified recyclables range from extra costs and system damage to previously recyclable materials turning unavoidably into non-recyclable garbage. Another study [2] analyses and reviews the connection between not-properly categorised recyclables and health problems of employees working at sorting-plants. Therefore, having a robot doing this work with only a small margin of error can be quite helpful and can also save time, money, energy, resources, and not least human life.

I was always interested in and passionate about robotics. For me an important aspect of robotics is building and designing mechanical and electrical devices and by this to create something that has a practical function: a machine or a mechanism that can perform the task it is taught to do way better than a human.

The thesis presents how this recyclable material sorting robotic arm could help to solve some of the issues on a smaller scale: Starting from the idea, the basic design of the arm itself, creating the mechanical parts and putting them together with the electronic components whilst continuously testing them. Finally, writing and testing the object recognition program which enables the arm to detect and sort the refuse.

Abstract

This paper presents the development of a robotic arm, capable of sorting out recyclable rubbish correctly. The documentation explains the process of creating the mechanical parts of the robot and assembling them with the electronics, furthermore, it discusses the process of training an object detection model from two different databases using deep learning, which enables a Raspberry Pi pocket computer's camera to recognize different kinds of trash. It presents the difficulties of programming the arm to reach out to the right distance and pick up the trash based on a monocular depth estimation. In addition, the thesis explores a wide variety of applications of such robotic arms and the possible, future improvements in this field, while discussing the motivation behind the project itself.

Earlier works

There have been previously created robotic arms, whose task was to sort out recyclable trash as well. We can usually see these kinds of arms in larger factories, which take the given type of rubbish off the conveyor belt and reorder it.

There is a similar project to the one presented here, which was created in 2019, by the name of "*Recycle Sorting Robot With Google Coral*" [3]. However, the final form of this robotic arm differs from the one discussed in this paper significantly. This previously created project uses a rather different approach with respect to how the recognition-program has been taught and how the arm is programmed to reach for the trash. The above-mentioned robotic arm's recognition software utilized TensorFlow Lite [4] for learning, which is Google's machine learning model (see for more detail about TensorFlow Lite section 3.2). It used solely the *trashnet* dataset, and hand-labelled the images' bounding boxes. This project used transfer learning and applied the *MobileNet SSD V2 (COCO) model* [5] (see for more detail about *MobileNet* and COCO sections 3.3, 3.4). The arm's movement is determined by the number of frames the camera sees the object at, and according to the placement of the object, the arm reaches out to a given distance.

Another work worth mentioning is a project created by two Stanford University students in 2016, called "*Classification of Trash for Recyclability Status*" [6]. The project utilized the Torch framework (see for more detail about Torch section 3.2). Their thesis explains the method of creating an eleven-layer CNN [7] (short for Convolutional Neural Network, more detail about neural networks in section 3.1) model, which classifies objects based on their recyclability relying on a self-taught database. The dataset that was created is called *trashnet*, which was later published, and will be used for the object detection model (more detail about the object detection model in chapter 3.).

The robotic arm presented in this paper, on the other hand, uses a framework called Pytorch [8] for the training of the recognition software, which is a machine learning framework based on the torch library, and can be utilized mainly for computer vision (see for more detail about Pytorch section 3.2). The final model this project uses is based on a training that is a combination of two datasets, the *taco* and the *trashnet* (see for more detail about datasets section 3.5). As a result, this arm is distinct from the machines taking the trash off a conveyor belt, would be also usable in a natural environment and potentially could pick up trash from any surface and carry it to a designated place. Another main difference between the two projects is the method used when reaching for the identified trash (see for more detail about object-localization section 3.7).

Chapter 1. - First Steps

1.1. The Design

After deciding on creating a robotic arm that would eventually be able to sort out recyclable materials, the first step was to decide how the mechanical body should be built, to enable it to carry out its task with the greatest precision possible.

To create the sketches of each part of the arm, the application *Sketchbook* [9] was used, in which the components were first created individually, then layered to together.

Originally, the plan was to have four servos altogether on the whole arm, two *TP SG90 micro servos* [10] and two *MG996R servo motors* [11], as shown on Figure 1.1. The *SG90* micro servos are servo motors which operate using plastic gears, therefore weigh only nine grammes (thus, they are often referred to as *9g* servos), but still have a surprisingly strong torque at *1.8 kgf*cm*. These servos are usually used in smaller robotics projects, where less torque is enough. The *MG996R* motors feature metal gearing, resulting in an exceptionally high stall torque of 10 *kgf*cm*. These servos both operate at 4.8V to around 6.5V, but the stall current the *MG996R* servos can draw peaks up to 2.5 A.

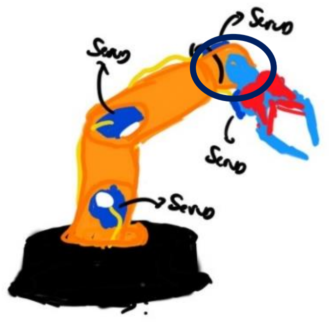

Figure 1.1.: The basic design of the arm. The third arm part is shown highlighted.

After researching other robotic arms, we realized that at least another smaller motor is needed to make the third arm part on the arm rotatable [Figure 1.1.]. Another option considered was at first to make the base part of the arm immobile, and to only move the other elements of it using servos. However, understanding, that this would limit its capability to pick up objects from all around, a new sketch was created, and the movement was planned to be done by a motor with the assistance of gears. Subsequently, it was decided to operate only an *MG996R* motor to move the base, as using electronics solely instead of extra gears offered a simpler and more convenient solution. Thus, at this stage, the arm was being moved by six servo motors. Table 1. shows the type of the servo used for each main joint. Figure 1.2. shows the second sketch made about the arm.

Table 1.

Main Joint:	**Servo used:**
Waist (base) part	*MG996R* servo
First arm part	*MG996R* servo
Second arm part	*MG996R* servo
Third arm part	*SG90* servo
Head-part rotation	*SG90* servo
Gripper	*SG90* servo

The table shows the servos used for each main joint of the arm.

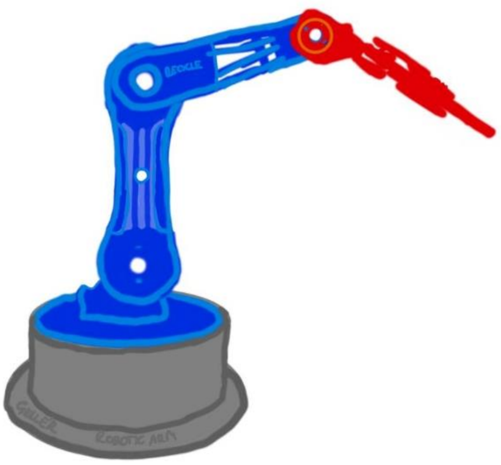

Figure 1.2.: Highlighting the important edges, this way, we can get a better view of all the given components.

1.2. 3D Modelling, Printing and Assembling of the Parts

The 3D designing of the components was done according to the plans created in *Sketchbook*, and the parts were remodelled based on *.stl* files [12] found on the internet. The STL format [13] is the most popular format for 3D printing of bodies, it is a neutral 3D file format and stores only geometrical information. The 3D designing of the objects was initially done in the CAD software Tinkercad [14], although later on Fusion360 [15] was utilized for the purpose of remodelling a part (for more detail see Appendix I). The STL files contained the parts of the arm divided into 10 different components. These components were the following:

- Base component
- Waist part
- First arm part
- Second arm part
- Third arm part
- Head part or gripper base
- The gripper, which consisted of two gears, four grip links, and two gripper-fingers.

The gripper was adjusted to allow it to grab onto objects better [Figure 2.1.]. The other parts were modified, with the aim of not using that much plastic when printing them, with the added benefit of certain parts of the arm becoming lighter. The waist part [Figure 2.2.] has been remodelled, cutting out additional plastic.

A “Recycle” caption was added on the side of the second arm part, and the component was optimized, to make it lighter [Figure 2.3.].

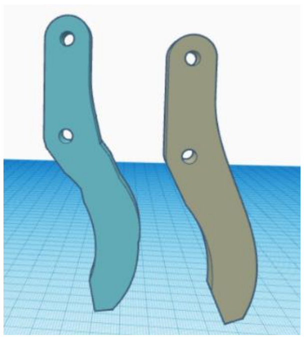
Figure 2.1.: Side by side comparison of the remodelled gripper finger (left) and the original one (right)

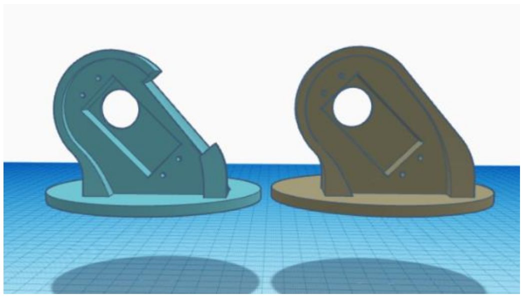
Figure 2.2.: Side by side comparison of the remodelled waist part (left) and the original one (right)

The first arm part's redesign included a perpendicular hole in the middle of the component to allow the tips of the servo cables to come out [Figure 2.4.]. Two cogs were added to the gears in order to achieve a larger angle of movement for the gripper [Figure 2.5.].

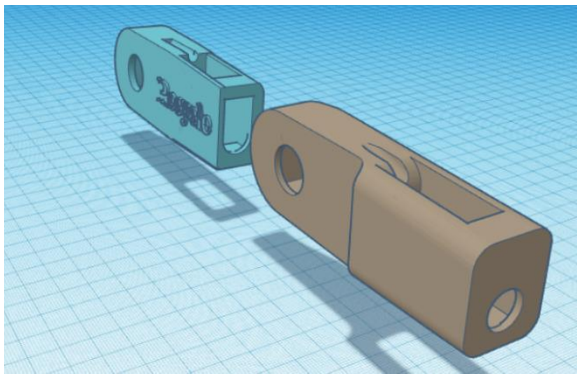
Figure 2.3.: The remodelled part (left) was optimized to use less plastic than the original one (right)

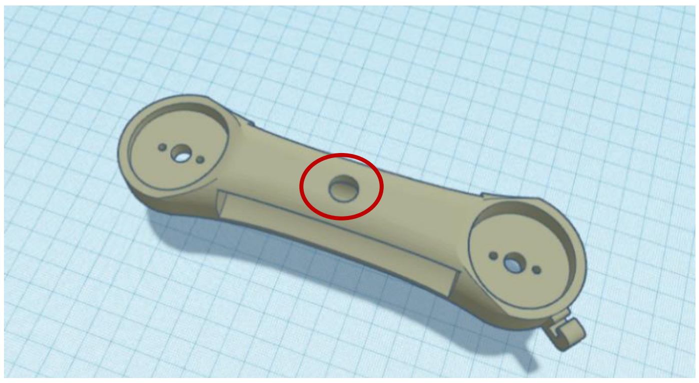
Figure 2.4.: The first arm part with a hole in the middle to allow the ends of shorter servo cables to be led out of the part.

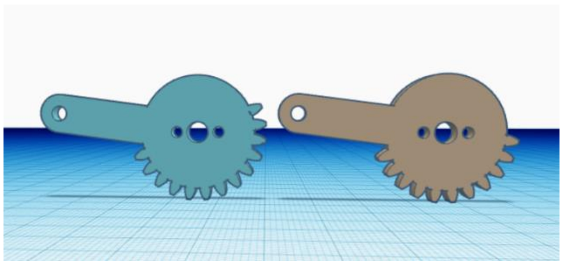
Figure 2.5.: The redesigned gear (left) compared to the original one (right)

The parts were redesigned one by one, and after all components were finished, they were assembled in Tinkercad to form the 3D version of the arm.

Finishing up the design, two types of rectangles have been added to the virtual model, each with the size of the given servo (*9g* or *MG996R*) and were placed in their designated places.

Figure 3. depicts the final 3D model of the arm. The components were printed from a black, PLA type plastic filament.

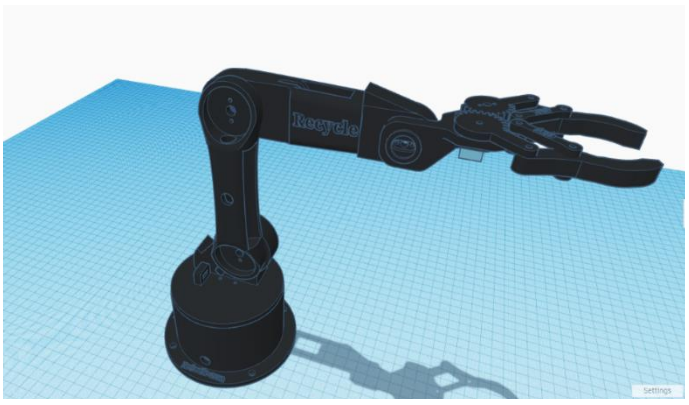

Figure 3.: The final 3D model of the arm

The servos required to assemble the 3D printed parts were acquired, along with the other electronics, and the assembling was done according to the sketches, models, and previous work.

The first step was assembling the printed components. (for more detail on how certain parts were assembled see Appendix I). Figure 4.1. shows the waist part and the base assembled.

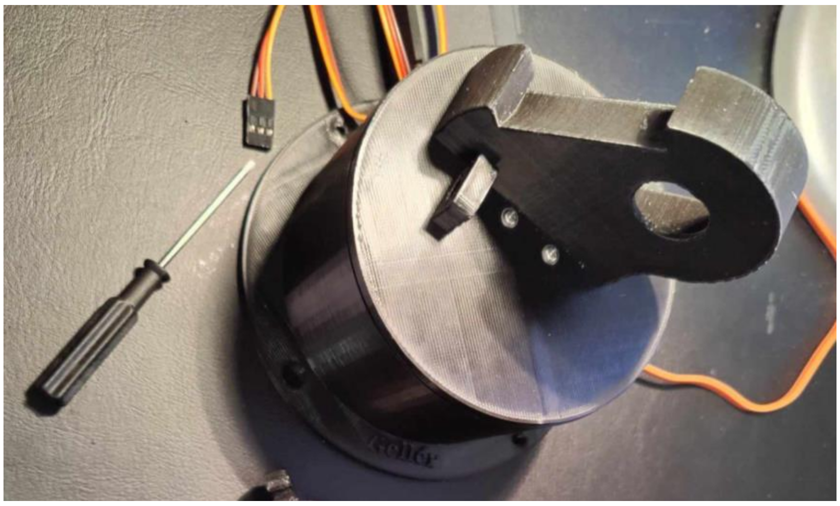

Figure 4.1.: The waist component secured on the top of the base part.

Next, another *MG996R* motor was added, this time to the waist part, which would carry the whole arm's weight. As the servo needed extra support, a rubber band was attached, connecting the waist component and the first arm part [Figure

4.2.]. Adding another circular head to that servo, the first arm part was screwed onto it. The following step was to assemble the second arm part with the first one.

Subsequently came the smaller, *9g* servos. The first servo was screwed inside the tip of the second arm part. The end of the motor was sticking out at the tip of the second arm part, and a horn was screwed on the tip of the servo, the third arm part was secured onto that. The second *9g* servo was put into the third arm part, with its circular end also hanging out on the other side, so the head of the gripper could be secured onto it [Figure 4.3.].

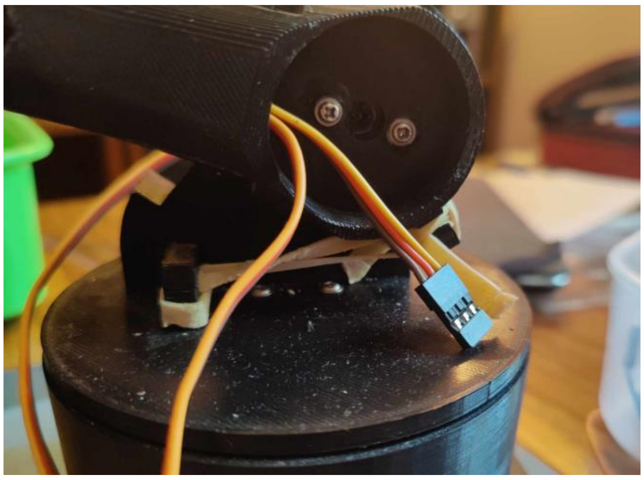
Figure 4.2.: The rubber band attached to the first arm part and the waist, so, it can support the movement of the servo.

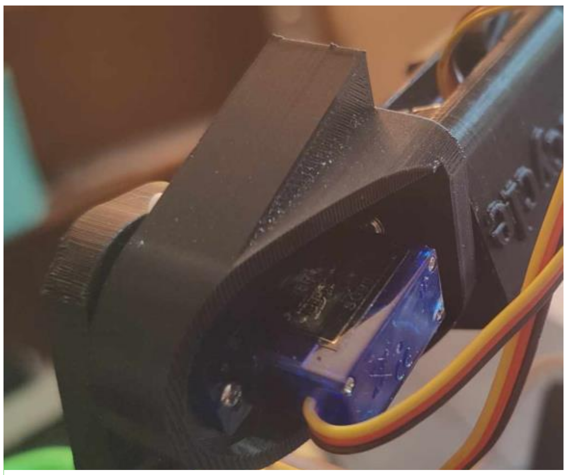
Figure 4.3.: The third arm part

A horn was placed on the servo, which first held the head part [Figure 4.4.], then another servo was screwed on it and with the help of that the two gears were pieced together [Figure 4.5.]. After securing the grip links to the head with bolts, the gripper-fingers had to be assembled with the gears and the grip links.

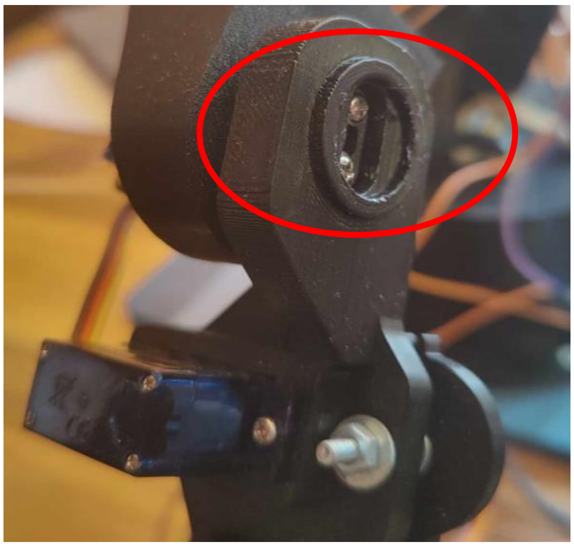
Figure 4.4.: The head part hanging from the third arm part

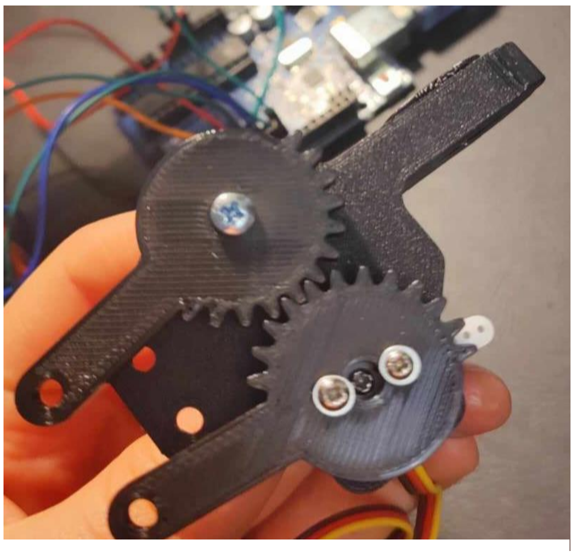
Figure 4.5.: The half-assembled head part

The last step was to put the arm on a foundation, to provide better stability for it when the servos move suddenly and must pick up an object of larger weight. A plastic board was used for this. The base was screwed onto the board [Figure 4.6.].

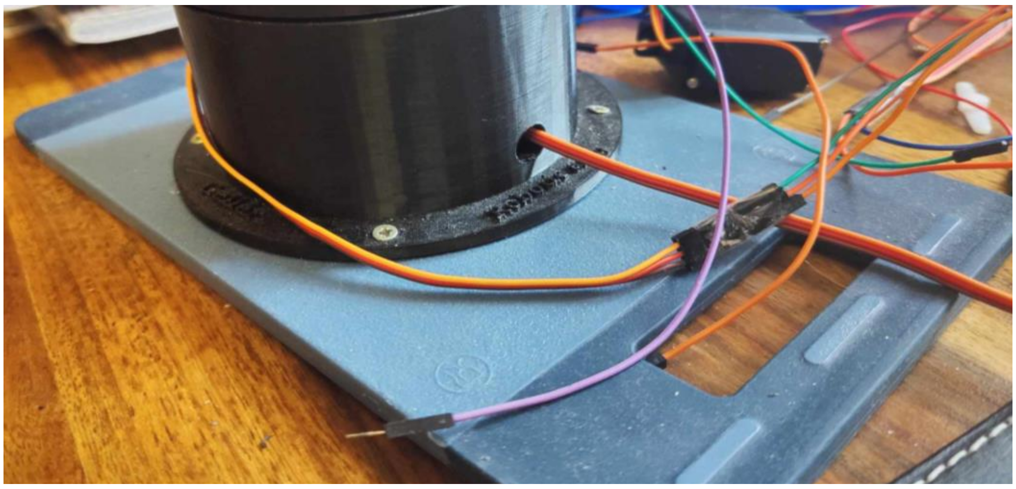
Figure 4.6.: The foundation of the arm

Figure 4.7. shows the arm after all the mechanical components were assembled.

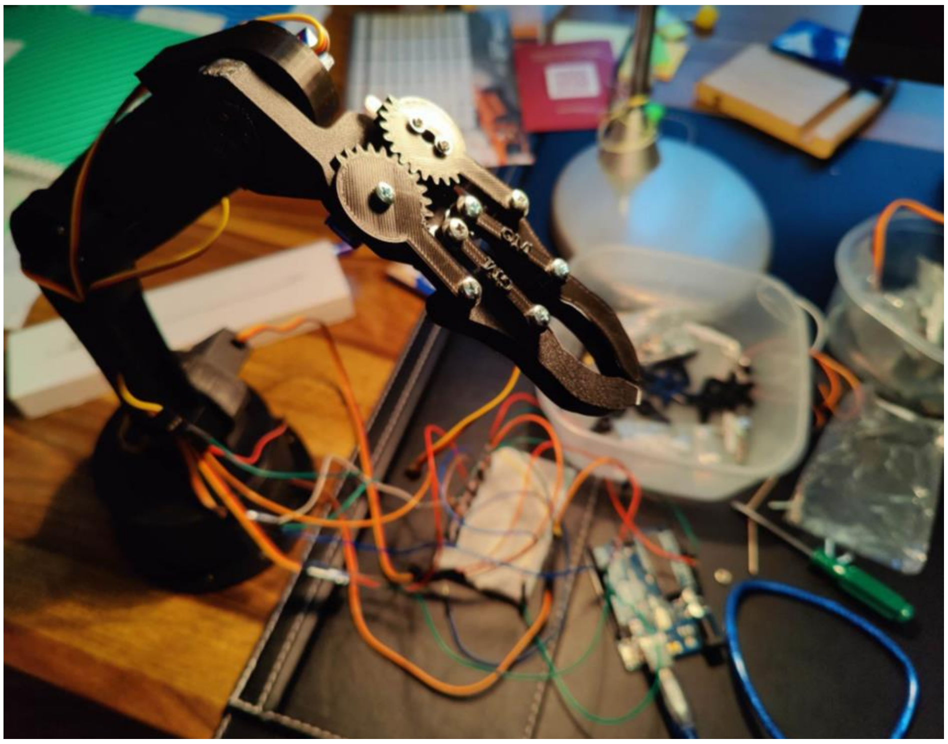
Figure 4.7.: The Robotic Arm, fully assembled.

1.3. The Hardware – Electronics

The basic circuit responsible for moving the servos is controlled by an Arduino Uno R3 circuit board [16]. The Arduino Uno is a microcontroller board, which has 20 digital input/output pins overall. The Arduino sends or receives the signals to or from the connected electronic components through these pins. Six of these pins can be used as PWM [17] (short for Pulse Width Modulation) outputs and 6 can be used as analogue ("analog") inputs. Certain digital pins (namely: 3, 5, 6, 9, 10, 11) are equipped with Pulse Width Modulation, or PWM, which is a technique for getting analogue results through digital means. Naturally, digital pins can only output serial data, meaning an "ON" or an "OFF" state, a "HIGH" or a "LOW", or in a numerical form, a "0" or a "1" state. PWM modifies this, as it allows the digital pins to output analogue data, therefore the pins are not restricted to outputting only two values but can output an infinite number of values within a continuous range. Installing certain libraries for the use of given components allows them to be used from any digital pin, without having to worry about PWM. For instance, the *Servo* library allows servo motors to be controlled from any of the 13 digital pins, even though the values that are sent to them from the Arduino are between 0 and 180 (the range of degrees a servo can turn in).

Furthermore, the Arduino can establish USB connection, it has a power jack, a reset button, and a bootloader [18] software pre-installed. The bootloader is the program that runs every time the reset button is pressed, or a new sketch is uploaded from the computer. This software is responsible for writing programs into the memory of the board. The Arduino R3 is the third, latest version of the Arduino Uno circuit boards. It contains everything needed to support the microcontroller; it can be programmed by simply connecting it to a computer through the USB port.

The other core part of the hardware is a Raspberry Pi 4 Model B [19]. The Raspberry Pi is similar to a normal desktop computer. Some of the main differences between the everyday PCs and Raspberrys are for instance the 40 GPIO pins the pocket computer has, the two camera ports which support the usage of the Raspberry camera add-ons, and that Pi 4 is powered through one USB-C port, with an adapter which outputs ideally 5 volts and 3 amperes. These small computers also have their own operating systems and all the information stored on a Mini-SD card, which allows simple data-transfer. The Raspberrys have their own version of Linux, the Raspbian OS. Raspberry Pis are often used in robotics projects, where it is crucial to have a computer that can fit in tighter places, is easily transportable and can give out commands to electrical components through its own pins, like servos, or even a complete circuit board. Using a Raspberry Pi for this project proved quite convenient, as it allows the arm to be controlled solely from a box of electronics and used together with the Raspberry Pi Camera V2 [20], which can be easily secured onto the head part of the arm, and thus object detection becomes possible. Figure 5.1. shows the Raspberry Pi connected with the camera, schematically.

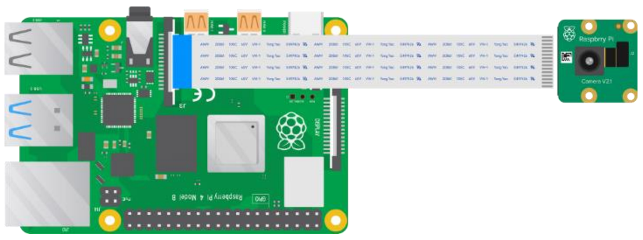

Figure 5.1.: A virtual representation of the Raspberry Pi 4 connected with the Raspberry Pi Camera V2 [21]

Table 2. shows each additional electric component that was used within the circuit.

Table 2.

Electric components	**Short description**
Servos	The motors responsible for moving the arm at each main joint.
A breadboard [22]	A simple, yet very useful extension to Arduino circuits. Breadboards allow cables to connect with other wires to form shorter paths between pins.
AC adapter	An outer power supply outputting sufficient current for the servo motors to operate simultaneously.
Raspberry Pi Camera V2 [Figure 5.2.]	The Raspberry Pi Camera is a custom designed add-on module for the Raspberry Pi hardware. The camera has a 5-megapixel resolution.
Argon Mini fan [Figure 5.2.]	A mini fan that was placed on the Raspberry Pi to keep it on operating temperature

The table shows each additional electric component that was used within the Arduino circuit or together with the Raspberry Pi.

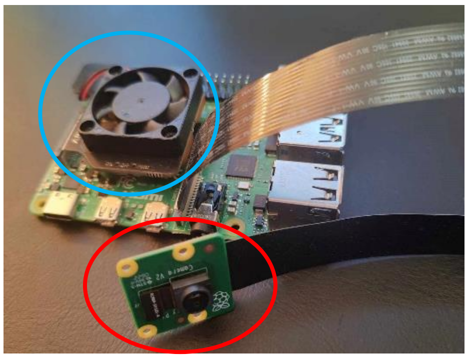

Figure 5.2.: The Raspberry Pi connected with the camera (highlighted in red) and the Argon Mini fan (highlighted in blue)

The *MG996R* servo motors need at least 4.8V of voltage and 2.5A of current to operate properly, which can't be supported from a computer's USB port. A computer's USB port can usually provide between 0.5 and 0.9A of current [23]. Therefore, an AC adapter had to be used, which is able to supply 6V of voltage and around 8A of current overall, to allow sufficient supply of energy for the motors to move. Figure 5.3. shows the completed circuit diagram, Figure 5.4. shows the same diagram schematically, and Figure 5.5. shows the Arduino circuit next to the mechanical body of the arm.

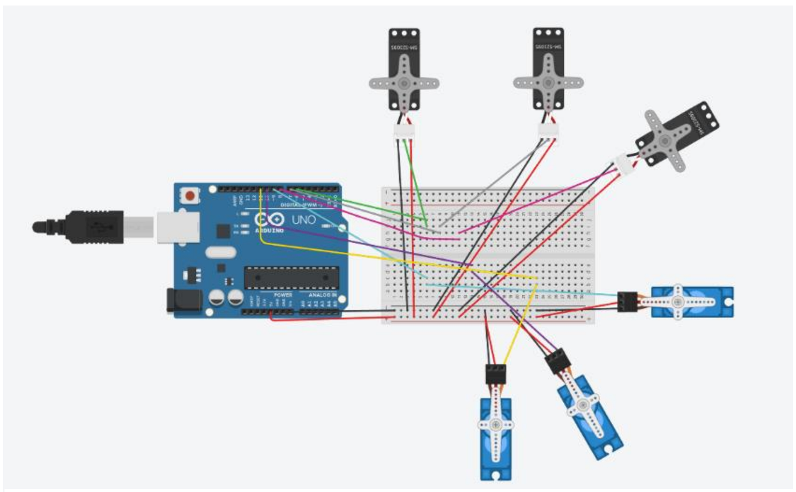

Figure 5.3.: The Arduino circuit diagram. The blue servos are the SG90 micro servos, and the black ones are MG996R motors.

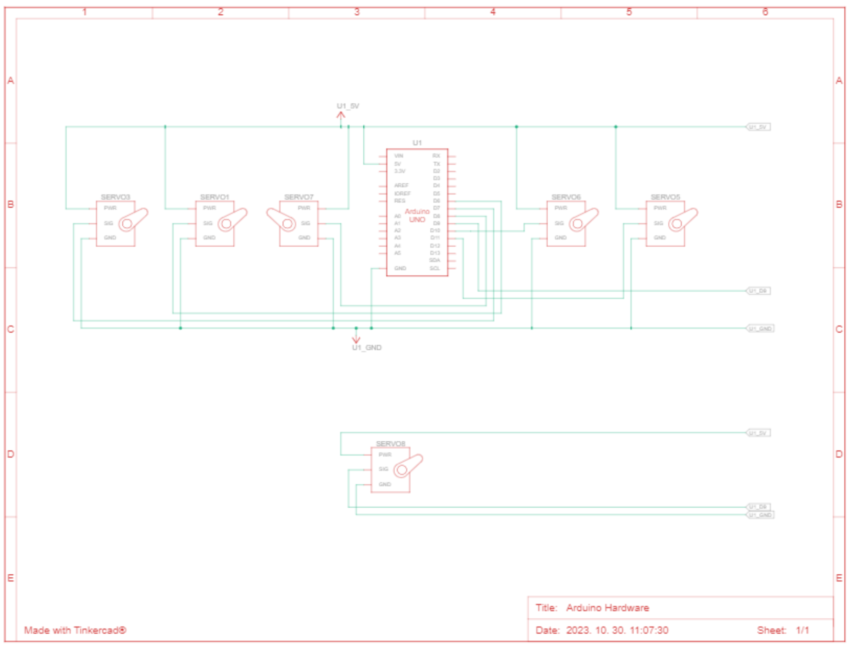

Figure 5.4.: The schematic diagram of the Arduino circuit

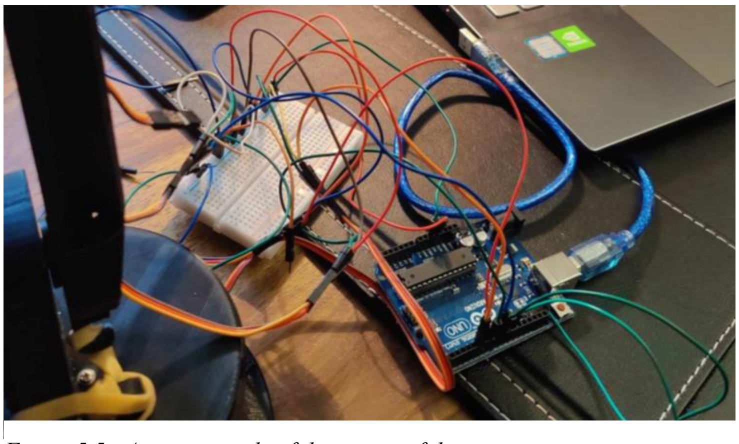
Figure 5.5.: A picture made of the wiring of the arm

The Arduino and the Raspberry Pi have to be connected in order for them to function together properly. The Arduino is the circuit board commanding the servos, and the Raspberry Pi will be providing the data necessary for the Arduino to be able to identify which servos to rotate. Since the Raspberry is just like a normal computer, the Arduino can be easily connected to it using a cable. This connection allows the devices to communicate via Serial communication. The Arduino will move the servos according to what the Raspberry Pi outputs.

Chapter 2. – Testing the Circuit

2.1. The Code

Following the completion of the mechanical body of the arm, the testing phase began. For the tests, codes were written in Arduino's official coding software, Arduino IDE [24]. The Arduino language is quite similar to the well-known coding language C++ [25], although it contains an addition of special methods, libraries, and functions, which are required for the smooth control of electronic components. The INO language has a simpler syntax, hence, it is easy to understand, and it also has built-in Serial communication, which is a necessity in certain cases. For instance, when the circuit board has to send information to a computer, communication via Serial is the most evident method. Regardless, the Arduino circuit boards can be programmed by a wide variety of other coding languages, including Python [26], C++, and C [27], although the use of these languages requires more than a novice's understanding of coding in general.

After connecting the servos to the Arduino, a code that was written in Arduino IDE gave the circuit board orders, to move one of the servos back and forth, from 0 to 180 degrees, with a 1 second delay in between. As mentioned previously, the Arduino IDE contains various libraries. The library required for the servos to move properly and for them to be controllable from non-PWM pins as well is called *Servo* library, which is easily downloadable inside the software. Therefore, the library was imported with just one line of code. After that the servos, then the pins that the motors were connected to were defined. (For more detail on testing the servos, see Appendix II).

The *TP SG90* micro servos have a stall current of around 650mA, and an operating voltage of around 4.8V, which the computer is able to support through its USB port. The servos were moving correctly, according to the angles outputted by the Arduino. The gripper was adjusted as well. In the closed state, the maximum weight of the object the gripper is able to grab and hold on to is around 50 grams, which is sufficient for picking up smaller objects, such as aluminium cans and plastic cups.

2.2. Challenges with Moving Several Servos Simultaneously

When an Arduino has to control more than six servo motors at the same time, or a stepper motor [28], a so-called *motor shield* is required to aid the Arduino. Arduino *shields* [29] are board-extensions, which easily integrate with the Arduinos, and offer a wide variety of specialized capabilities, such as controlling more motors with the previously mentioned *motor shield*, or *sensor shields*, which are engineered specifically for the use of certain sensors, or even a *GIGA display shield*, which allows a touchscreen solution specifically for Arduino Mega boards.

Changing the code allowed the Arduino to send out signals to move more than one servo at a time through the pins. The following table [Table 3.] shows the servos moved along with the results of the various test-cases.

Table 3.

Servos Moved:	**Observation:**
Base, first arm part	No recognizable delay or poor performance
First arm part, Second arm part	Some inaccuracies in the timing of the movement (no delay is taken), despite that it functions properly
Base, Second arm part	Some inaccuracies are also spottable in the movement of the Second arm part
Second and Third arm parts, gripper-head	Functions as expected; however, the third arm part always moves a bit during the delay, when it should not
First and Second arm parts, gripper head	Some delay is recognizable in the movement of the first and second arm parts
First and second arm parts, gripper head, gripper	Delay is recognizable in the movement of the first and second arm parts, the gripper functions properly

The table contains observations on the movement of the servos during testing.

Moving more servos simultaneously did not seem to cause any serious problems, however, the *MG996R* servos stopped from time to time or rotated with more delay than specified, which was a cause for concern [2. row of Table 4.].

Several other difficulties arose throughout the creation of the mechanical body of the arm and the circuit. Table 4. summarizes these issues, and briefly presents the solutions to each and every one of these problems (for more detail on the problems and solutions see Appendix III).

Table 4.

	Issue:	**Cause:**	**Solutions:**
1.	Servos losing their functionality before the testing phase	Short-circuited the Arduino at some point	Changing the Arduino boards, updating the Arduino IDE, changing some of the electronics
2.	Poor performance of the *MG996R* servos	Insufficient power supply	Changing the AC adapter
3.	Servos not getting power from the outer power supply	Wrong connections in the circuit	Changing the connections in the circuit
4.	The servo was not able to move the second arm part	The gripper head and second arm part were too heavy	Adding a rubber band, reprinting certain components from a lighter plastic

The table briefly shows each hardware-related problem that was faced after the completion of the mechanical body of the arm.

The performance problems observed in Table 3. and shown in Table 4., second row, were solved by changing the power supply, as the previous one was not able to provide sufficient current and voltage for the servos to operate simultaneously.

Chapter 3 - The Object Recognition Model

3.1. The Approach – How Should the Program Detect Trash?

The main approach was to create an object recognition model that would eventually be able to detect trash, and then sort it into different groups based on their types. To create such models, a neural network [7][30] was utilized.

Artificial neural networks are machine learning models, which consist of artificial neurons. Neurons are parameterized by adjustable weights and organized into multiple layers. The network takes images as inputs which are converted into real numbers based on the RGB values of the pixels. Subsequently, the network performs various computations using the inputs and weights (e.g., convolutions) and produces bounding box coordinates and labels as outputs. These predictions are compared with the known boxes and labels from the training images and the discrepancy is quantified as an error. During training, the network weights are optimized in order to decrease the error on the training dataset.

The trained network can then be used to provide predictions on previously unseen, arbitrary images. Figure 6. is a representation of how this neural network learned, with an example image from the *taco* dataset (see for more detail about datasets section 3.3).

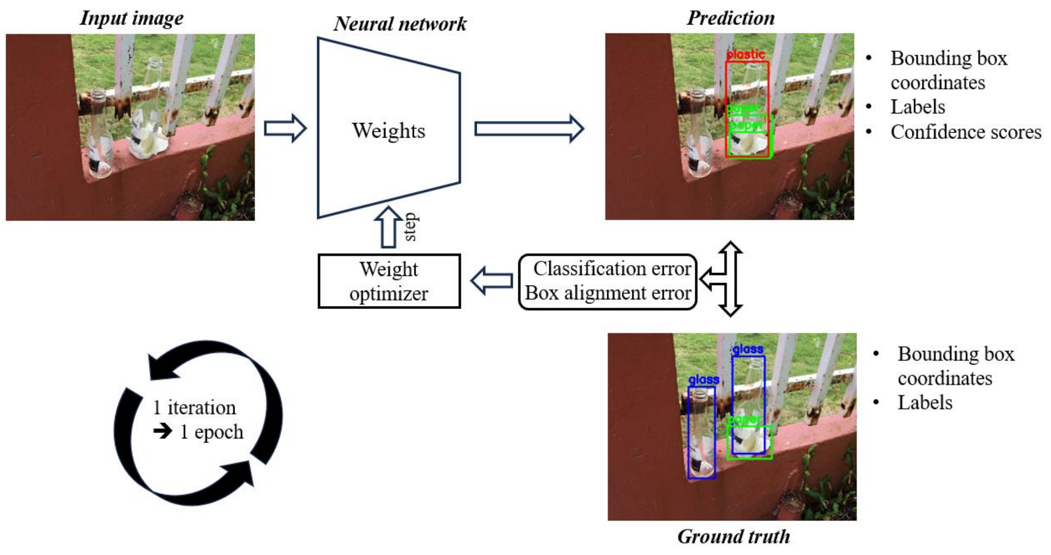

Figure 6.: The learning curve of this neural network

The number of epochs a model was trained for represents the number of iterations, or how many times the model saw the whole dataset.

The coding language that was used throughout the making of the object detection model was Python. Python is a high-level, beginner-friendly, and general-purpose programming language. Python was the obvious choice when it came to selecting the programming language to create the object detection model with, as it is the standard language used in machine learning projects, therefore a lot of websites offer great support, and Python is compatible with most libraries and extensions which are necessary to make the project work.

3.2. Frameworks

Throughout the creation of the object recognition model three frameworks were used. One of these utilized at the start of the project was TensorFlow Lite. TensorFlow Lite is Google's machine learning model, an open-source deep-learning framework. It is a collection of tools to optimize TensorFlow models [31] to run on mobile devices. TensorFlow was developed in 2015 and is a variety of open-source software libraries for AI and machine learning, which utilize deep neural networks. The difference between the two models is that TensorFlow Lite provides the ability to perform predictions and to recognize objects based on an already trained model and it is a lighter version of the original model, designed specifically for mobile computing platforms.

As we ran into difficulties with the use of TensorFlow Lite (for more detail see Appendix IV), the framework we decided to use afterwards was Pytorch, which is another framework used for machine learning. Pytorch also provides access to state-of-the-art deep learning models for various computer vision tasks e.g., object detection. Moreover, these models are also available pre-trained on large image databases.

Video streams were handled using the OpenCV framework [32], which is a general computer vision library, including all the necessary functionalities for camera handling, image conversion, pre-processing, and visualization.

3.3. Gathering Data

To be able to create an object recognition model with the help of Pytorch, which detects and recognizes recyclable trash, we had to look for a large enough database, so the model would be able to distinguish between different kinds of recyclables with a small margin of error. The dataset we found and decided to use was *taco* [33]. Taco (short for Trash Annotations in Context) is an open image dataset, which contains images taken of litter in nature, but has pictures of rubbish from other environments as well. The annotations provided to the *taco* images are JSON files, which is a compact and convenient format for storing class labels, super categories and object bounding boxes. COCO (short for Common Objects in Context) [34] is known as a large database for object detection and segmentation, containing over 330,000 images of everyday items. The similarity between the two datasets comes down to the format used, which is JSON format in both cases, which allows a more convenient usage of the databases.

Table 5. represents the common attributes of JSONs in object-labelled datasets [34] (such as COCO and *taco*).

Table 5.

Information	General information about the dataset, such as version number, date created, and contributor information
Licenses	Information about the licenses for the images in the dataset
Images	A list of all the images in the dataset, including the file path, width, height, and other metadata
Annotations	A list of all the object annotations for each image, including the object category, bounding box coordinates, and segmentation masks (if available)
Categories	A list of all the dataset object categories, including each category's name and ID

The table shows the attributes that the files with JSON format include.

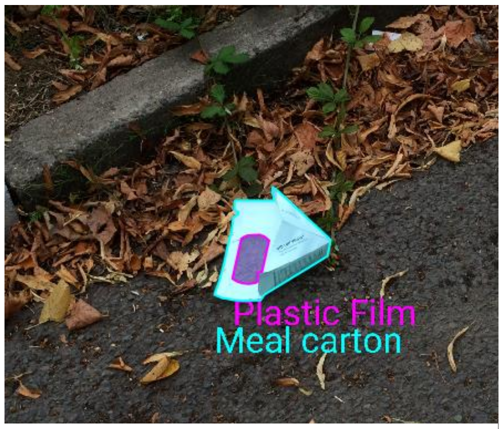

Figure 7.: An annotated image from the taco dataset

The *taco* dataset contains 4613 images and for every image there is an annotation used to identify the picture, there are labels, which are used to classify objects, and a bounding box, indicating where the rubbish is located on the picture. Figure 7. shows an example image from the *taco* dataset of a “Meal carton” and a “Plastic Film”. Both litters are located and labelled on the image. On this picture, the object is segmented, meaning that the pixels on the image are classified based on whether they belong to the object or not. Segmenting an image is a more difficult task, but for this object recognition model it was not necessary, placing trash in bounding boxes was equally functional and proved more convenient, as it requires less memory to operate than image segmentation would.

Inside the dataset each image is divided into a super category, and inside that super category, into a normal category. For instance, a picture of a plastic cup is in the super category ‘Cup’ and inside that, in the (normal) category ‘Plastic cup’. The *“Categories.json”* works similarly to a Python dictionary of the classes, meaning the *taco* dataset distinguishes between 60 categories overall (for all the categories see Appendix IV).

3.4. Object Detection Using the *Taco* Categories

For the object detection, the *SSD MobileNet V3* [35] model was selected, which was pre-trained on the COCO database, and we fine-tuned it on the *taco* dataset using Pytorch. The *SSD* (short for Single Shot Multibox Detector) *MobileNetV3* model is an object detection neural network designed specifically for mobile devices, therefore it only requires a small amount of memory to work from, which would allow the object detection to run smoothly from a Raspberry Pi as well. The reason we chose to use this model was that it has quite low-memory requirements compared to other CNN models, while its performance and accuracy is still sufficient.

Another considered approach was not to fine-tune the whole network, but only the last few layers of the model, where the weights were almost set to the appropriate values, and the model did not have to "learn" that much. This approach would have taken less computational resources, but ultimately was rejected, as the predictive performance of this training was not as promising as retraining the whole model on another dataset.

The script used for retraining the *SSD MobileNetV3* model on the *taco* database (*"training.py"*) can be found in Appendix IV. Originally, we would've used a cloud to train the *taco_190_full* object recognition model solely from this database, however the training was done on the Informatics faculty's computer, on an NVIDIA Titan XP graphics card, which made the process smoother.

Tests on the first model indicated clearly, that a retraining was necessary in order to reach the desired results. Figures 8.1. and 8.2. show examples of such test-cases. On figure 8.1., a paper cup is recognized as "Plastic film" (not a Plastic cup), and the empty aluminium can on Figure 8.2. is detected as "Food waste" and "Foam food container".

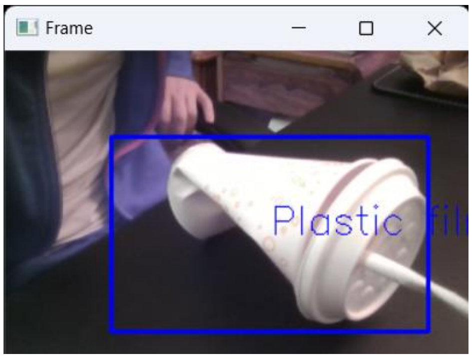

Figure 8.1.: The paper cup detected as plastic film using the taco_190_full model

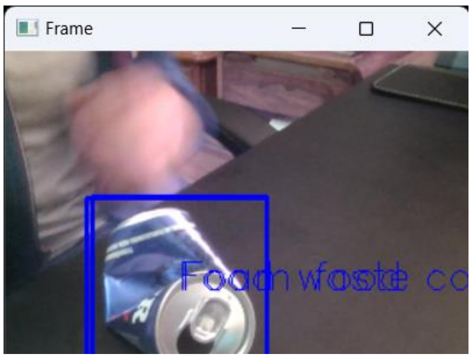

Figure 8.2.: The aluminium can is recognized as food waste or/and "Foam food container with the taco_190_full model

The first model was trained for 190 epochs, which was insufficient, therefore we decided to increase this number up to 500 for training the second model, which simply meant that the program would have more chances to distinguish between objects on the images. Creating the second model took longer, than training the previous one. Following the training of the second model, the testing of the object-detection began.

3.5. Testing

Since object detection models are hard to evaluate in general, moreover, the size of the dataset is limited, we only performed testing on a case-by-case basis. In particular, a few tests have been made after each and every greater step. We also tested the default recognition model, which was the previously mentioned *MobileNetV3* baseline model pretrained on the COCO database, which was not fine-tuned on *taco*. The default model was then imported into the code, and therefore we were able to test the labelling and the bounding capabilities of the framework. At first, the model was only able to detect objects present in the basic database, such as couches and humans, showing their assigned labels and drawing a rectangle around the perceived location of the bodies.

Following, the next testing took place after the training of the model was successfully completed. The model's trash recognition capabilities were tested by showing different household recyclables to the camera, from metal soda cans to plastic bottles and paper cups. The results of the tests clearly indicated that the model needed a larger set of data to work from and a longer time for training to be more precise, as it often misrecognized objects, for instance paper cups for plastic films, or the coloured side of an aluminium can for some kind of plastic or food waste, or it was not able to decide where the object was located, therefore the bounding box was misplaced inside the frame. Whilst using the model, even random objects have been recognized as rubbish of some sort, and even then, the wrong class of trash. Changing the angle or the distance did not resolve this issue.

The second model has been trained solely from the *taco* trash database as well, but now for more epochs, theoretically leading to better predictive performance. Giving the program more chances to be able to identify the objects with a smaller margin of error did not show significant change in its performance. The object recognition model was still unable to distinguish between a tissue and a plastic film, or between a plastic cup and an aluminium can. Figures 9.1. and 9.2. represent some of the *taco_full_500* model's recognitions.

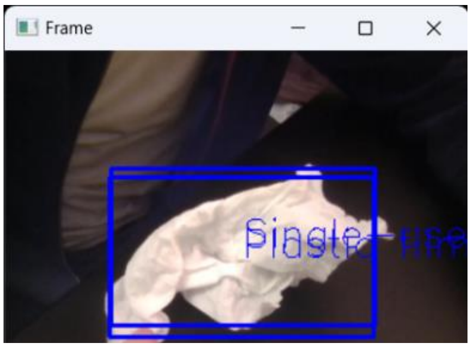

Figure 9.1.: A crumpled tissue recognized as a "Plastic film" by the taco_full_500 model.

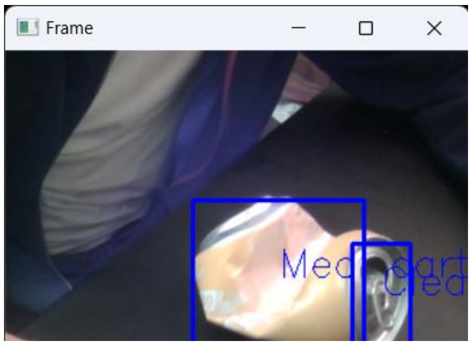

Figure 9.2.: An aluminium can recognized as two different rubbish: a "Meal carton" and a "Clear plastic bottle".

A different approach was needed, because it was clear, that the dataset of *taco* in itself won't be sufficient enough for detecting the correct type of trash.

3.6. Object Detection Using the *Trashnet* Categories

After concluding, that the *taco* dataset won't be able to fulfil the needs of the object detection model in itself, we had to come up with a different solution. We concluded, that we couldn't just retrain the previous model with more epochs, meaning, we couldn't just show the whole dataset to the model more times during the training, because over-increasing the epoch number can lead to an overfit model which will function with only a small margin of error on the data it was trained on, but on every other kind of data the margin of error will be significantly higher. We agreed on the fact, that the root of the problem was the dataset itself, as it did not have a sufficient number of images to work from. One possible solution would have been to look for extensions to this dataset, which might have not existed, and another one would've been to create our own pictures and label them by hand, which would've taken a lot of time. Instead, we decided to approach the issue from a different angle.

Another dataset used for machines sorting recyclable rubbish is the *trashnet* dataset. This dataset contains less images than *taco*, with only 2525 pictures to work from overall. The images did not meet our expectations, with regard to the environment they were made in. The *trashnet* dataset has images made in front of a white background, which allows for a convenient localization of the object on the image, however, it does not grant support for when the arm would have to identify trash in a natural, messy, or crowded environment, as this dataset was mainly created for machines that take trash off the conveyor belt, but still, we decided to use it, as it would still add value to the model combined with *taco*'s images. The categories these images were sorted into were based on their materials as a litter, for instance cardboard and glass. We checked the bounding boxes of certain images, and concluded, that some boxes were drawn around the whole picture, meaning, certain rubbish took up the whole frame. This proved to be an unfortunate situation, as the camera won't zoom in on the images in a way that the trash will fill out the whole picture, but it will observe it from a distance. Therefore, it won't be able to distinguish that precisely between certain recyclable rubbish, which could lead to issues with the arm's sorting capabilities. Figures 10.1. and 10.2. show the differences between when the trash fills out the whole frame and when it is observed from a distance. The same piece of cardboard is not recognized by the model on the second picture (this test was done using the first iteration of the model combining the two datasets).

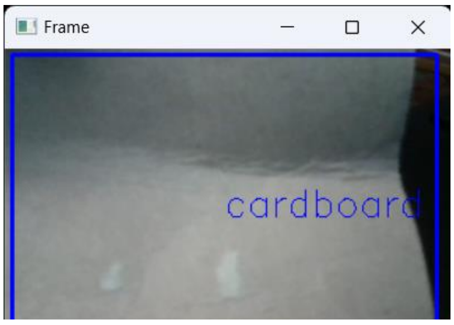

Figure 10.1.: A piece of cardboard filling out the whole frame of the camera.

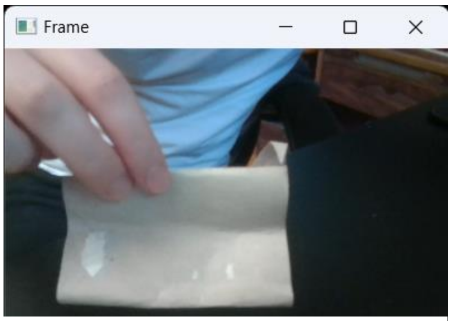

Figure 10.2.: The same piece is not recognized when observed further away from the camera.

We wanted to unify the two datasets, while keeping only six categories, which the *trashnet* images would be sorted into, and the plan was to assign the pictures from the *taco* dataset to these classes as well. This was done by assigning each *taco* category to one *trashnet* category in a text file. In the first row there was a number indicating the category. Table 6. represents the trashnet categories.

Table 6.

Key	Value
"0"	glass
"1"	paper
"2"	cardboard
"3"	plastic
"4"	metal
"5"	trash

The table shows to which key which kind of material was assigned to.

Using this text, the images of the *taco* database were recategorized, and then the *trashnet* dataset's images were added as well.

The models were trained systematically, there are ten models overall, each with a different number of epochs, ranging from 50 to 500, where the model which saw the whole dataset five hundred times has better predictive performance.

Testing this model consisted of running it and then showing it different kinds of litter, items quite similar to the ones the object-recognition model was trained on. The first few tests gave positive feedback, the model trained for 500 epochs was more accurate in identifying recyclable trash than the previous models, although it wasn't perfect. Oftentimes the model detected the whole frame as an object, which will possibly lead to issues with the pick-up mechanism later. This is due to the fact that many bounding boxes in the trashnet database take up the whole frame, as mentioned previously (especially cardboard). Even though the recognition model was more accurate than the previous ones, further testing was necessary, to decide whether the program had to be retrained up to 1000 epochs to theoretically increase accuracy, or not, as it currently seems as there are no other datasets which prove useable for the object recognition model. The following figures show test-cases made with the last iteration of the model (500 epochs).

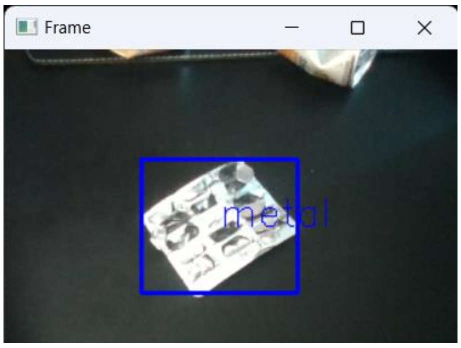

Figure 11.1.: The model can detect shining metal surfaces quite well.

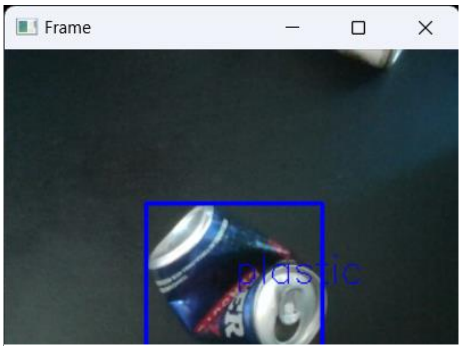

Figure 11.2.: The aluminium can is still detected as plastic.

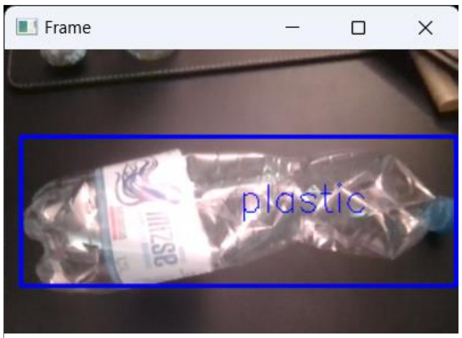

Figure 11.3.: The model is able to recognize the plastic bottle.

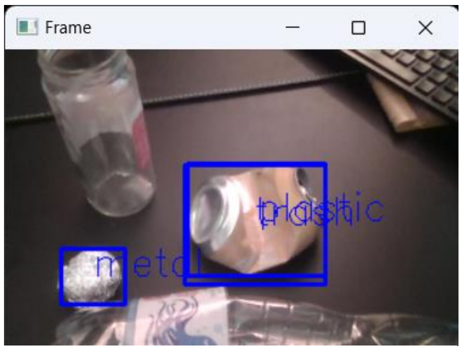

Figure 2.: The model recognizes the metal rubbish, but it cannot distinguish between the categories "trash" and "plastic" for the metal can.

3.7. Object Localization Via Monocular Depth-estimation

We concluded that a reasonable approach would be for the arm to use a depth-estimation model to check the distance that the trash is located at. Following, the arm would deduct the distance the object is positioned at according to the values the measurement outputs or according to the body's colour on the depth-estimated image.

The project uses a depth-estimation model called *MiDaS* [36]. *MiDaS* depends on a library called *timm* [37], which is a deep-learning library and a collection of state-of-the-art computer vision models. *MiDaS* has 3 types of models, a low accuracy and low memory-usage one, there is a high accuracy model which uses the most memory, and one in between the two other models, with a medium size and accuracy. The plan is that the Raspberry will be running the model with medium accuracy, but this is quite dependent on how much RAM the other interfering programs (such as the object detection itself) will use. The model can be loaded in with just a few lines of code, although, to carry out the original idea -which was to run a depth-estimation on a picture captured by the camera the moment it detects trash- a new code has to be written, connecting the depth-estimation program with the object detection model and the movement of the servos. The task is to move the motors according to the distance of the trash on the picture. The way this would work is that the depth of the picture the camera captured would be estimated, and then the object would be located on the image. Then, the estimated depth of the object would be taken from a pixel (or a depth-value which from the surface of the object), and to every different value the depth-estimation outputs, a metric distance is assigned. This would require the calibration of the model [38], which we leave to be done as future work.

This procedure would allow the arm to reach out to the right distance after spotting rubbish. Figure 12.1. shows an original picture from the *trashnet* dataset and 12.2. is the depth-estimated version of the image. The object is clearly visible on the estimated picture.

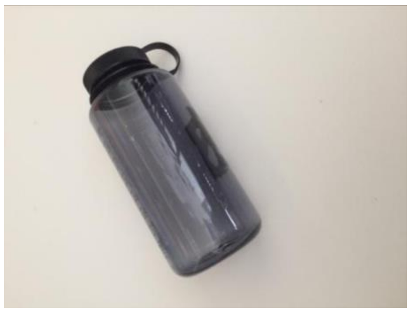
Figure 12.1.: A plastic bottle from the trashnet dataset

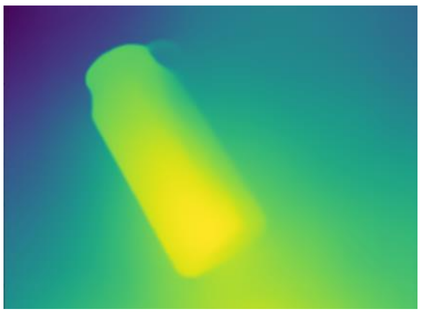
Figure 12.2.: The depth-estimated version of the image shown on Figure 12.1

However, the depth-estimation *MiDaS* provides is not always completely accurate. Figures 13.1 and 13.2. are an example, where we can see, that the (darker) shadow of the glass bottle was estimated to be closest to the camera, although we can easily deduce, that the point closest is the bottom edge of the bottle. In this case, the software could not decide whether the shadow is part of the glass bottle, as the bottle's colour almost perfectly matches the shadow's. This mistake would not have a serious effect on how far the arm would reach, but on a larger scale it could manipulate it to move the wrong servos a wrong amount.

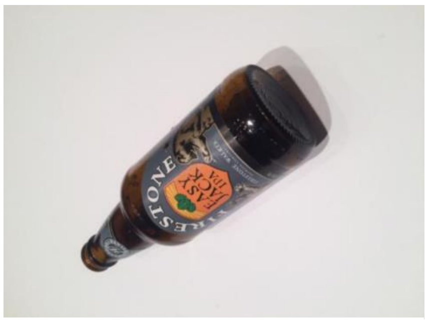
Figure 13.1.: A glass bottle from the trashnet database

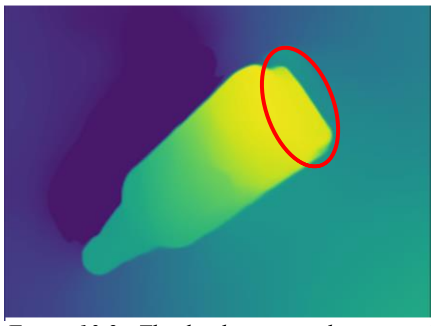
Figure 13.2.: The depth-estimated version of the glass bottle on Figure 13.1. Highlighted is shown the estimation of the shadow, which was estimated to be closest. This is a mistake of the depth estimation.

Figures 14.1. and 14.2. show an example of depth-estimation that was executed on a picture from the *taco* database. It is clearly visible that there are two objects and there are three spots where the yellowish/bright green colour is noticeable, meaning, there are three places which are equidistant from the camera. This example provides overview of why it is necessary to locate the trash on the depth-estimated picture, because if the arm was to only look for the object closest to the camera, it would not always reach for the trash, but for any other object that interferes within the frames. Therefore, the method to be used will consist of taking the depth value from the centre of the bounding box.

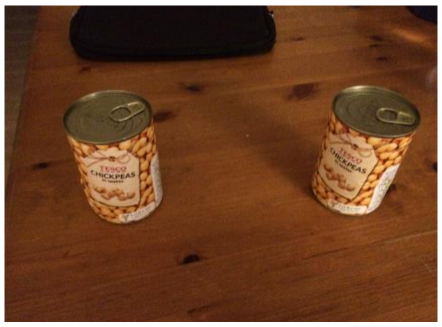
Figure 14.1.: An image of metal cans from the taco dataset

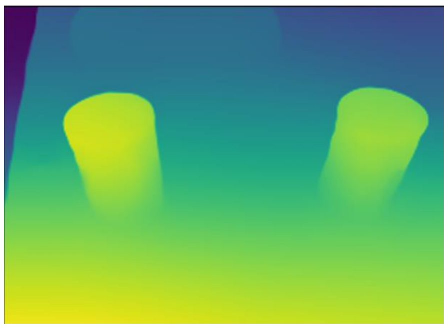
Figure 14.2.: The depth-estimated version of Figure 14.1.

One other considerable possibility would be estimating depth on a real-time video. This option was rejected for the time being, as this procedure would require a large amount of free memory from the Raspberry Pi to function, which won't be necessarily provided, as running the object detection with the camera operating whilst the Raspberry Pi is also giving out instructions to the Arduino to control the servos would overwhelm the device.

Chapter 4. - Future improvements

4.1. Future Development of the Arm

The arm still awaits its completion, the final stage would be to finish combining the depth-estimation model and the pick-up mechanism. The plan is, to have the camera do a depth-estimation once it spots litter, and then identify the rubbish on the depth-analysed picture. Then, according to that, the arm would classify the object into one of many "distance-groups", meaning for instance if the value of the depth of a pixel which is located on the object is small (according to calibration), the servos should be moved a small amount in that direction. Completing this mechanism proves difficult, as it states issues that are still unresolved.

Another method which is considered provides a different approach from when the trash is spotted. The key point of the new "theoretical" method is drawing vectors on the frame, from the centre of the frame towards where the centre of the object's bounding box is located at. Then the vectors can be broken down to x-axis and y-axis components. Moving a certain amount on a certain axis would require the arm to move only one of its servos at a time. The closer it gets, the smaller the vector gets, and once it is in the centre of the frame, the arm would have to adjust according to the object's vertical distance (z-axis component, which could not be shown on the frame) using the values taken from the depth-estimation, or just move the head of the gripper downwards, if the arm is already over the object, and then grab it, in theory. Figure 15. shows an example of how this method would look like on the camera frame.

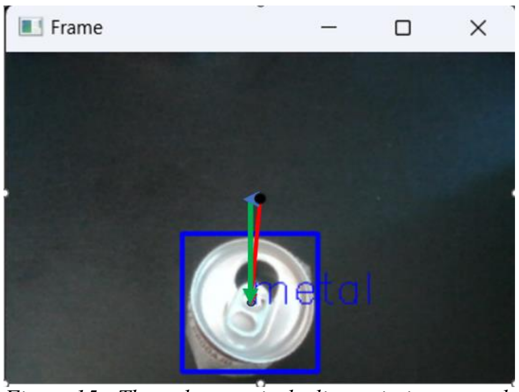

Figure 15.: The red vector is the line pointing towards the middle of the bounding box from the centre of the frame, the blue vector is the x-axis component of the red vector, and the green vector is the y-axis component.

This method is still highly theoretical and has its weak points, but with some work it might replace the original approach.

Following, if the previously mentioned vector-location approach is rejected, a path of improvement of the pick-up mechanism would be to add more distance groups over time, in hopes of achieving better accuracy.

In addition, another path of development that is considered for the arm is to add wheels to the base, which would allow it to go around and scan in certain environments, then pick up any trash it has detected. For this, the arm would have to be recoded, and we would also need motors and a shield for the Arduino to be able to control the wheels. This possible upgrade of the arm states a lot of questions and problems, but it would be a great way to improve its efficiency.

Finally, another way to further develop the robotic arm would be to make it remote-controlled and to create an interface for it. This would allow a user to control the arm remotely from a smartphone, the rubbish it picked up could be analysed/checked, and the arm could be controlled to move around as an RC car (having the wheels built on it). This upgrade would allow the arm to not only sort and pick up recyclable materials, but also to be controlled by a human being to perform other tasks, for example reaching into a narrow place or carrying certain objects.

4.2. Areas of Further Improvement on this Field

Nowadays almost everything has become automated, and the waste-industry should be no exception either. A study from 2006 [39] argues, that "*Automated sorting for plastic recyclables has been seen as the way forward in the plastic recycling industry.",* which serves as a great example of how similarly automation of the recycling industry was perceived merely seventeen years earlier. The thesis is documenting the creation of the prototype of an Automated Plastic Sorter System, using image processing, rather than expensive state-of-art technologies, such as X-ray and Infra-Red monitoring. Although, the sorting of recyclables using machine learning is not mentioned in it.

There is still plenty of room for improvement on this field, but a lot has changed since the beginning of the century. Recyclable sorter robotic arms have become rather popular in waste-sorting factories, as they can be much more efficient than human workers, if trained properly. Another study [40], from 2021 discusses that "*workers generally leave after a few months. Another challenge is that waste management companies are struggling to recruit workers [...]*" and then brings the argument *"Waste separation can happen by hand [...] or else automatically separated in the MRF (short for Materials Recovery Facilities). Sensor-based sorting techniques have been broadly used lately as an alternative to manual sorting of solid waste. Contrasted with the manual sorting, the material segments obtained are [...] of higher purity, and economic value."* This study, along with several others [41] show the contrast between manual and automated waste-sorting. The technological advancements of the past few years would allow facilities to switch, sooner or later, to fully automated sorting of recyclable trash, not only in certain advanced sorting facilities, but all around the world.

We can conclude that the AI powered sorting of litter is not yet industrialised, but chances are, it will be sooner rather than later, thanks to the rapid development of artificial intelligence and machine learning. Automating the waste industry has countless benefits, from saving resources and time to saving human lives, with the technology available nowadays, it is certainly an option to be considered.

Contribution

The robotic arm, including the electronic components was sketched, designed, 3D printed and assembled by Márton Lukács Gellér, with the help of the cited publications, studies, resources. Data for the object recognition program were gathered from the referenced websites and pages, and the object detection models used in this project were trained and tested by Márton Lukács Gellér and Dr. Bence Márton Bolgár. The original object localization technique was suggested by Dr. Bence Márton Bolgár. The thesis was written by Márton Lukács Gellér.

Acknowledgements

This work would not have been possible without the support of my alma mater, the Piarista High School of Budapest, and especially my teacher Gergely Botond Kiss, who have generally been supportive of my goals, provided continuous inspiration for research and free initiative and also offered technical support, when needed. I am thankful to the school's supportive community, especially Márton Ferenc Pitter, who supported this work with his technical resources.

I would also like to thank Ákos Vida, for supporting this work in practical aspects.

I am especially indebted to Dr. Bence Márton Bolgár who is my supervisor, and who has given me extensive personal and professional guidance throughout the project.

Finally, I am grateful to my family for their unwavering support and encouragement.

References

[1]: Team, R. (n.d.). *What Happens When Waste Isn't Sorted Properly?* [online] knowledge.recycle-smart.com. Available at: https://knowledge.recycle-smart.com/blog/properly-sorted-waste.

[2]: Poulsen, O.M., Breum, N.O., Ebbehøj, N., Hansen, Å.M., Ivens, U.I., van Lelieveld, D., Malmros, P., Matthiasen, L., Nielsen, B.H., Nielsen, E.M., Schibye, B., Skov, T., Stenbaek, E.I. and Wilkins, K.C. (1995). Sorting and recycling of domestic waste. Review of occupational health problems and their possible causes. *Science of The Total Environment*, 168(1), pp.33–56. DOI: https://doi.org/10.1016/0048-9697(95)04521-2.

[3]: Bernas, A. (December 23, 2019). *Recycle Sorting Robot With Google Coral*. [online] Available at: https://www.hackster.io/bandofpv/recycle-sorting-robot-with-google-coral-b52a92

[4]: Abadi, M., et al. (2015). *TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems*. [online] Available at: https://static.googleusercontent.com/media/research.google.com/en//pubs/archive/45166.pdf.

[5]: Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, Liang-Chieh Chen (2018). *MobileNetV2: Inverted Residuals and Linear Bottlenecks*, IEEE/CVF. Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4510-4520, DOI: 10.1109/CVPR.2018.00474

[6]: Yang, M. and Thung, G. (2016). *Classification of Trash for Recyclability Status*. [online] Available at: https://cs229.stanford.edu/proj2016/report/ThungYang-ClassificationOfTrashForRecyclabilityStatus-report.pdf.

[7]: Schmidhuber, J. (2015). Deep Learning in Neural Networks: An overview. *Convolutional Neural Networks*, pp.9–22. doi: https://doi.org/10.1016/j.neunet.2014.09.003.

[8]: Paszke, A. et al., (2019.) PyTorch: An Imperative Style, High-Performance Deep Learning Library. In Advances in Neural Information Processing Systems 32. Curran Associates, Inc., pp. 8024–8035. [online] Available at:

http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

[9]: The application used can be found here: https://www.sketchbook.com/

[10]: Imperial College London, PC, M. (2014). *SERVO MOTOR SG90 DATA SHEET*. [online] Available at: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf.

[11]: components101.com. *MG996R Servo Motor Datasheet, Wiring Diagram & Features*. [online] Available at: https://components101.com/motors/mg996r-servo-motor-datasheet.

[12]: How To Mechatronics. (n.d.). *Arduino Robot Arm STL Files*. [online] Available at: https://howtomechatronics.com/download/arduino-robot-arm-stl-files/

[13]: Anon, (2020). *Top 3D File Formats for 3D Commerce, Social & More | VNTANA*. [online] Available at: https://www.vntana.com/blog/demystifying-3d-file-formats-for-3d-commerce-and-more/.

[14]: The website of Tinkercad: https://www.tinkercad.com/dashboard

[15]: The website of Fusion360: www.autodesk.com/products/fusion-360

[16]: Arduino (2022). *UNO R3 | Arduino Documentation*. [online] docs.arduino.cc. Available at: https://docs.arduino.cc/hardware/uno-rev3.

[17]: *About PWM:* https://www.bequiet.com/hu/insidebequiet/1688

[18]*: Arduino bootloader documentation:* https://docs.arduino.cc/hacking/software/Bootloader

[19]: *Official Raspberry Pi 4 Model B documentation*: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/specifications/

[20]: *Official Raspberry Pi Camera V2 documentation:* https://www.raspberrypi.com/documentation/accessories/camera.html

[21]: The figure can be found here: https://www.arducam.com/raspberry-pi-camera-pinout/

[22]: tangentsoft.com. (n.d.). *What is a 'Breadboard'?* [online] Available at: https://tangentsoft.com/elec/breadboard.html.

[23]: He, Fan. (2015). *USB Port and power delivery: An overview of USB port interoperability*. pp. 1-5. Doi: 10.1109/ISPCE.2015.7138710.

[24]: https://docs.arduino.cc/software/ide-v2

[25]: https://cplusplus.com/reference/

[26]: Van Rossum, G. & Drake, F.L., 2009. Python 3 Reference Manual, Scotts Valley, CA: CreateSpace.

[27]: Prinz, Peter; Crawford, Tony (December 16, 2005). C in a Nutshell. O'Reilly Media, Inc. p. 3. ISBN 9780596550714.

[28]: Stepper motors: uk.rs-online.com. (n.d.). *Everything You Need To Know About Stepper Motors | RS*. [online] Available at: https://uk.rs-online.com/web/content/discovery/ideas-and-advice/stepper-motors-guide.

[29]:https://guides.temple.edu/c.php?g=419841&p=2908632

[30]: Wang, S.-C. (2003). Artificial Neural Network. *Interdisciplinary Computing in Java Programming*, 743, pp.81–100. doi: https://doi.org/10.1007/978-1-4615-0377-4_5.

[31]: TensorFlow: tf.keras.layers.Dense - TensorFlow API Documentation, https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense (2023. jún.)

[32]: Bradski, G., (2000). The OpenCV Library. Dr. Dobb&#x27;s *Journal of Software Tools.*

[33]: Proença P., Simões P. (2019). *Taco (Trash Annotations in Context)*. Fonline] Available at: http://tacodataset.org/

[34]: Lin, T.-Y., Maire, M., Belongie, S., Bourdev, L., Girshick, R., Hays, J., Perona, P., Ramanan, D., Zitnick, C. and Dolí, P. (2015). *Microsoft COCO: Common Objects in Context*. [online] Available at: https://arxiv.org/pdf/1405.0312.pdf.

The webpage summarizing the JSON attributes in the COCO dataset: https://www.v7labs.com/blog/coco-dataset-guide

[35]: Howard, A., Sandler, M., Chu, G., Chen, L.-C., Chen, B., Tan, M., Wang, W., Zhu, Y., Pang, R., Vasudevan, V., Le, Q.V. and Adam, H. (2019). *Searching for MobileNetV3*. [online] arXiv.org. Available at: https://arxiv.org/abs/1905.02244.

[36]: MiDaS: Ranftl, R., Lasinger, K., Hafner, D., Schindler, K. and Koltun, V. (2020). Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-shot Cross-dataset Transfer. *arXiv:1907.01341 [cs]*. [online] Available at: https://arxiv.org/abs/1907.01341.

[37]: Wightman, R. (2019). *timm: (Unofficial) PyTorch Image Models*. [online] PyPI. Available at: https://pypi.org/project/timm/.

Doi: 10.5281/zenodo.4414861

[38]: Farooq Bhat, S., Birkl, R., Wofk, D., Wonka, P. and Müller, M. (2018). *ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth*. [online] Available at: https://arxiv.org/pdf/2302.12288.pdf.

[39]: Abd Wahab, Dzuraidah & Hussain, Aini & Scavino, Edgar & Basri, Hassan. (2006). Development of a Prototype Automated Sorting System for Plastic Recycling. American Journal of Applied Sciences. 3. DOI: 10.3844/ajassp.2006.1924.1928.

[40]: Erkinay Ozdemir, M., Ali, Z., Subeshan, B. and Asmatulu, E. (2021). Applying machine learning approach in recycling. *Journal of Material Cycles and Waste Management*. doi:https://doi.org/10.1007/s10163-021-01182-y.

[41]: Uzosike, Canice & Yee, Lachlan & Padilla, Ricardo. (January, 2023). Small-Scale Mechanical Recycling of Solid Thermoplastic Wastes: A Review of PET, PEs, and PP. Energies. 16. 1406. DOI: 10.3390/en16031406.

Appendix I. – Modelling and Assembling the Parts in Detail

During the redesigning phase, the sizes of the holes on the parts were changed, to match the diameter of the screws that were used, which were M4*30 mm bolts and nuts, and the screws which were included with the *MG996R* and *SG90* servos (two different sizes). Bolts were acquired, which were necessary to assemble certain parts of the arm, one of these was the gripper and the gears moving it.

The redesigning was done according to the sketches, and the parts were remodelled based on the *.stl* files [See section 1.2.]. Before redesigning, a hand-made sketch was created, to highlight the parts that required modifications [Figure A1].

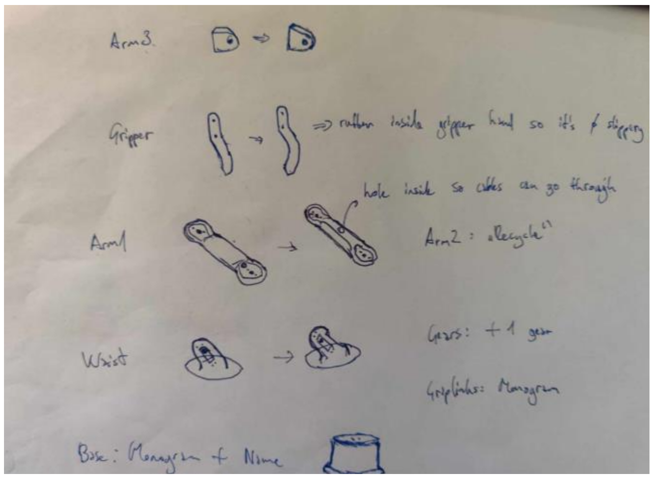

Figure A1.: A sketch representing certain parts and the necessary modification to each of these components.

Other modifications to the original files included adding small captions on the base and grip links.

Another 3D part was also created and printed, which was responsible for securing the camera onto the head part. This model was created in Fusion360, after measuring certain parameters of the camera, and the sides of the servo which it would be placed on. Figure A2.1 depicts this part's virtual model, and Figure A2.2 shows this part already printed and placed on the arm.

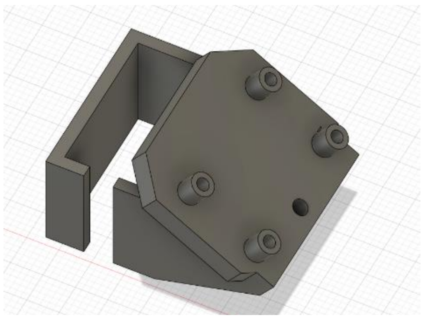
Figure A2.1.: The camera holder's 3D model

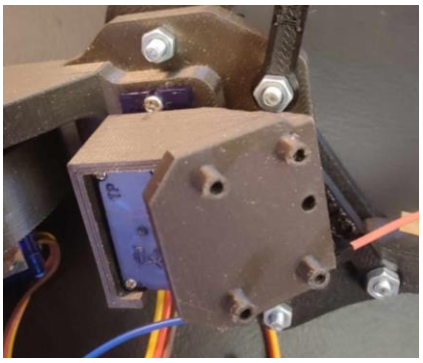
Figure A2.2.: The camera holder assembled with the servo located on the head-part.

Starting off the assembling process at the base, an *MG996R* servo was screwed into the bottom part, facing upwards, so the top of the bottom part could be put on it.

At first, it seemed that the screws were not fitting in the printed holes, but shortly after, it became obvious that they only needed a stronger twist, so the thread could carve its way in the plastic. A circular horn was added on top of the first servo, and then the waist part was secured on top of that. In the original model of the waist part that was found on the internet, an extra pin was designed to hold a rubber band, that would help the servo to move, which was not changed during the redesigning.

After adding the first arm part with another *MG996R* servo, the step that followed differed from the previous ones. The horn of the servo had to be screwed on the other end of the first arm part first, then a servo had to be added to the second arm part separately, following, the second arm part was assembled with

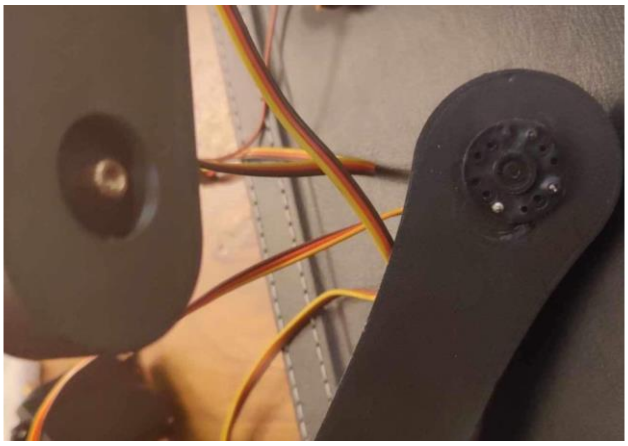
Figure A3.: The third arm part with the servo's tip hanging out (left) and the horn screwed on the first arm part (right)

the end of the first component through popping the horn onto the servo's end. [Figure A3].

In addition, the cables were led through the first arm part's hole, so they weren't disturbing the assembling process. Subsequently, a servo was screwed inside the tip of the first arm part [Figure A4]. Then, a horn was screwed on the tip of the servo, and the third arm part was secured onto that, using two more bolts [Figure A4, highlighted].

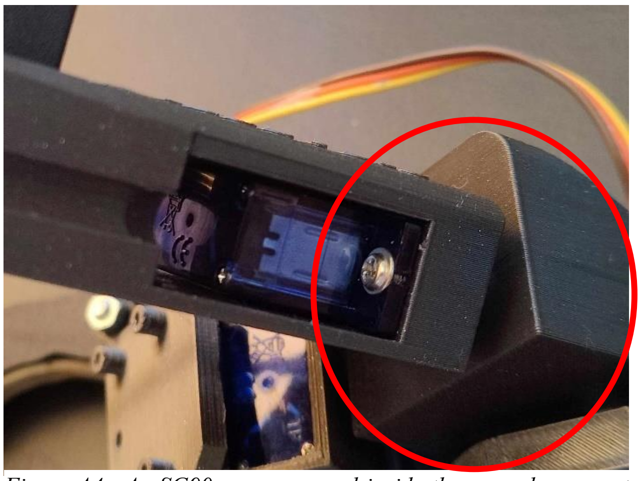

Figure A4.: An SG90 servo screwed inside the second arm part with the help of two servo screws, on the tip of that came the third arm part [highlighted]

Before moving on to assembling the head part (gripper) of the arm, it was constructed separately, and after concluding the correct method for piecing its parts together, it was built onto the third arm part, which first held the head part on that, there was another blue servo and with the help of that the two gears were pieced together [For more Figures see section 1.2].

Appendix II. – Moving the Servos – Tests in Detail

At first, only one long cable was used to connect the servos with the breadboard, and from there another one to the desired pins on the Arduino. It was necessary to change this, so the robot would have greater freedom and space to move in. Therefore, cables were acquired and plugged together, making the initial wires longer. This caused the servos to slow down, and in certain scenarios completely stop moving, therefore we had to switch back to using only one cable to connect the servo's pins with the breadboard. We concluded that poor connection between the cables or slow data transfer due to having more wires connected was the cause of the problem. Later on, we managed to fix this, and was able to use multiple cables with some of the servos.

First, the angles that the motor would rotate were inputted. The code was verified, and then uploaded. The motor responsible for turning the arm around from the base worked well on the first try. Following, we disconnected the first motor and plugged the second servo into the Arduino. The rubber band we added to help support the movement of the second *MG996R* servo, which was carrying the whole weight of the arm and would eventually carry the whole weight of the object picked up as well, was adjusted, and the code was executed. The only task left was to adjust the range of the angles the gripper's servo would move in. On the first try, we made it turn from 0 to 180 degrees. This caused the servo to overstrain the plastic cogs of the gears. Therefore, the servo was disconnected, and the script was changed, so the servos would only have to move from 0 to 90 degrees. This range of motion permitted the servo to move freely, and the gripper could open fully without any issues. we defined 0 degree as the "closed" state of the gripper, and 90 as the "open" one.

As the first test-case for moving multiple servos, the simultaneous movement was tested by moving the base 90 degrees, and at the same time, the first arm part from 0 to 120 degrees. This seemed to work, however, we were able to spot some delay in the movement of the first arm part, which we decided to ignore, as it didn't seem relevant.

Subsequently, we experimented with moving some of the other joints simultaneously, only two at a time. After concluding that moving two motors at a time didn't cause any issues to arise, we commanded the Arduino to move 3 motors at once, then four. For the test-cases and observations see Table 3. of section 2.2

Appendix III. – Challenges in Detail

Following the completion of the mechanical body of the arm, the servos lost their functionality. We couldn't identify the source of the problem, but we were aware, that the laptop was not able to upload the sketch on the board, therefore, it was assumed, that something was wrong with the Arduino Uno. Experimenting with different solutions, switching the board enabled us to start moving the smaller servos.

Changing the boards once didn't quite solve these problems in the long run. The same uploading problems appeared for a long time after the microcontrollers were changed. We concluded, that at some point the Arduino was probably short-circuited, which could've damaged the processor or the bootloader [for more information on the bootloader see section 1.3]. Using the new board, the testing phase restarted, as the servos were first moved one by one, then together, and the arm seemed to be slowly gaining its functionality back. A hole was drilled at the tip of the first arm part to hold another rubber band which will support the movement of the second arm part, as the *MG996R* servo had difficulties lifting it. A third bolt was screwed into a hole that was drilled right next to the other screws holding the two arm parts together, to secure them together better [Figure A5].

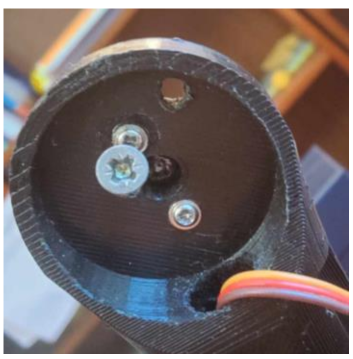
Figure A5.: A screw added for stability, and the hole for holding the rubber band

We faced a greater issue with the other large, *MG996R* servos. As these motors can draw up to 2.5A of current during movement, especially when moving under load, we had to make sure the adapter that was used was able to supply that. An AC adapter was bought, as the computers USB port wouldn't have been able to power all motors and an Arduino Uno alone. When plugging the Arduino on the power supply and the computer's USB port together, the servos were able to move, but they weren't operating properly for some reason. When we tried using only the power supply, the Arduino wouldn't start moving the servos. After some research, we concluded, that even though the power supply should've supported enough current and voltage both for the Arduino and the motos to operate, either it wasn't, or we connected the circuit the wrong way. Therefore, the connections were checked, and knowing that the outer power supply gives power through the Vin pin, not the 5V one, as the computer does, the circuit was rewired. Changing the order which the cables were connected in allowed the servos to finally operate properly, although their movement was still rather uncontrolled and slow. Therefore, we decided to use another AC adapter, which supplies more current and voltage than the previous one, and therefore makes sure the servos are not under-powered and can move precisely simultaneously.

Appendix IV. – Testing the Object Recognition Model

Certain difficulties arose during the usage of TensorFlow Lite. First off, the utilizable version of the framework was not compatible with our version of Python, therefore we had to download an additional kernel, allowing us to switch between versions. Using the compatible version of Python, TensorFlow Lite was still operating rather slowly, even though this was the lighter version of the original TensorFlow. Therefore, concluding this would slow the process significantly, we decided to switch to Pytorch.

Later on, after rewriting the *test.py* Python script, instead of showing a number, the labels were showing up as the name of the object the program was detecting.

For the testings, the next testing took place after the training of the model was successfully completed and we were able to import the *taco_full_190.pt* model into the *test.py* code.

The 60 *taco* categories, in JSON format: {"0": "Aluminium foil", "1": "Battery", "2": "Aluminium blister pack", "3": "Carded blister pack", "4": "Other plastic bottle", "5": "Clear plastic bottle", "6": "Glass bottle", "7": "Plastic bottle cap", "8": "Metal bottle cap", "9": "Broken glass", "10": "Food Can", "11": "Aerosol", "12": "Drink can", "13": "Toilet tube", "14": "Other carton", "15": "Egg carton", "16": "Drink carton", "17": "Corrugated carton", "18": "Meal carton", "19": "Pizza box", "20": "Paper cup", "21": "Disposable plastic cup", "22": "Foam cup", "23": "Glass cup", "24": "Other plastic cup", "25": "Food waste", "26": "Glass jar", "27": "Plastic lid", "28": "Metal lid", "29": "Other plastic", "30": "Magazine paper", "31": "Tissues", "32": "Wrapping paper", "33": "Normal paper", "34": "Paper bag", "35": "Plastified paper bag", "36": "Plastic film", "37": "Six pack rings", "38": "Garbage bag", "39": "Other plastic wrapper", "40": "Single-use carrier bag", "41": "Polypropylene bag", "42": "Crisp packet", "43": "Spread tub", "44": "Tupperware", "45": "Disposable food container", "46": "Foam food container", "47": "Other plastic container", "48": "Plastic glooves", "49": "Plastic utensils", "50": "Pop tab", "51": "Rope & strings", "52": "Scrap metal", "53": "Shoe", "54": "Squeezable tube", "55": "Plastic straw", "56": "Paper straw", "57": "Styrofoam piece", "58": "Unlabeled litter", "59": "Cigarette"}

The python file had to be rewritten, where the categories the model was sorting the items into was not the JSON dictionary containing the *taco* labels, but just a traditional python dictionary, with the keys as the indices of the recognizable materials, and the values as the class of the rubbish (metal, paper, etc.). For instance, when the output of the model would be “0”, the item pair with key “0” would be selected from the categories, and the corresponding value -which is glass in this case- would be showed above the rectangle drawn around the object in the camera frame, instead of the *taco* categories, which had the actual types of trash as labels in it.

The *training.py* code was relying on using the previously mentioned neural networks and weights, together with the chosen optimizer to train an object-detection model from a given database. The source-code is shown on Figure A6.

```
import numpy as np
import torch, torchvision
from matplotlib import pyplot as plt
import cv2
import json
from tqdm import tqdm
import os
import xml.etree.ElementTree as ET

num_epoch = 501
bs = 64
with open("Marci_taco_annot_640.json") as f:
    annot = json.load(f)

with open("Marci_taco_categories.json") as f:
    cats = json.load(f)

taco2trashnet = np.loadtxt("tacototrashnet.txt", dtype=int)

images = []
targets = []

for a in annot:
    im  = (cv2.imread(f"images_640/{a}")/255).astype(np.float32)[:,:,[2,1,0]]
    labels = annot[a]['labels']
    boxes  = annot[a]['boxes']
    trashnet_labels = [taco2trashnet[l] for l in labels]

    images.append(torch.tensor(im).permute(2,0,1))
    targets.append({"boxes": torch.tensor(boxes), "labels":
torch.tensor(trashnet_labels)})

trashnet_categories = {'metal' : 4, 'cardboard' : 2, 'paper' : 1, 'thrash' :
5, 'glass' : 0, 'plastic' : 3}

for file in os.listdir("thrashnettraining\Garbage classification\\train"):
    if ".jpg" in file:
        im = (cv2.imread(f"thrashnettraining\Garbage
classification\\train\{file}")/255).astype(np.float32)[:,:,[2,1,0]]
        tree = ET.parse(f"thrashnettraining\Garbage
classification\\train\{file[:-4]}.xml")
        root = tree.getroot()

        for bbox in root.iter("bndbox"):
            xmin = int(bbox.find("xmin").text)
            ymin = int(bbox.find("ymin").text)
            xmax = int(bbox.find("xmax").text)
            ymax = int(bbox.find("ymax").text)

        for n in root.iter("name"):
            label = trashnet_categories[n.text]

        images.append(torch.tensor(im).permute(2,0,1))
        targets.append({"boxes": torch.tensor([xmin,ymin,xmax,ymax]),
"labels": torch.tensor([label])})

for file in os.listdir("thrashnettraining\Garbage classification\\test"):
    if ".jpg" in file:
        im = (cv2.imread(f"thrashnettraining\Garbage
classification\\test\{file}")/255).astype(np.float32)[:,:,[2,1,0]]
        tree = ET.parse(f"thrashnettraining\Garbage
classification\\test\{file[:-4]}.xml")
        root = tree.getroot()

        for bbox in root.iter("bndbox"):
            xmin = int(bbox.find("xmin").text)
            ymin = int(bbox.find("ymin").text)
            xmax = int(bbox.find("xmax").text)
            ymax = int(bbox.find("ymax").text)

        for n in root.iter("name"):
            label = trashnet_categories[n.text]

        images.append(torch.tensor(im).permute(2,0,1))
        targets.append({"boxes": torch.tensor([xmin,ymin,xmax,ymax]),
"labels": torch.tensor([label])})
```

```
model = 
torchvision.models.detection.ssdlite320_mobilenet_v3_large(weights_backbone="D
EFAULT",trainable_backbone_layers=6,num_classes=60).cuda()

opt = torch.optim.Adam(model.parameters(),lr=0.002)
model.train()
with tqdm(total=num_epoch) as pbar:
    for i in range(num_epoch):
        idx = np.arange(len(images))
        nb  = int(np.ceil(len(images)/bs))
        np.random.shuffle(idx)

        sum_loss = 0
        for j in range(nb):
            batch = idx[j*bs:(j+1)*bs]

            opt.zero_grad()
            loss = sum(model([images[b].cuda() for b in batch], [targets[b] 
for b in batch]).values())
            loss.backward()
            opt.step()

            sum_loss += loss.item()

        if i%10 == 0:
            torch.save(model.state_dict(),f"taco&trashnet_full_{i}.pt")

        pbar.set_description("Avg loss: {:.3f}".format(sum_loss/nb))
        pbar.update(1)
torch.save(model.cpu().state_dict(),"taco&thrasnet_full.pt")
```

Figure A6.: The source code used for training the object recognition models. This is the last version of it, used for creating models from the taco and trashnet datasets.