Imperial College London

EE2 Group Project

FINAL REPORT

SortBot: Self-sorting Recycling Machine

Group 06

Group members:

Boskovic, Katarina (01064898)
Carrani, Jacopo (00944928)
Lee, Sze Tyng (01045611)
Liew, Guo Liang (01051242)
Lu Chen, Wendy (01069203)
Mathivanan, Abhinaya (01115427)
Rosu, Octavian (01061112)

Supervisor:

Dr Cong Ling

Submission date:
13.03.2017.

Table of Contents

1. Abstract

This project aims to tackle the issue of waste management in universities as current models of manual sorting of recyclables are tedious, time-consuming and discourage students from recycling more regularly. Market analysis and primary research in the form of surveying Imperial students was conducted. The technical solution proposed addresses this problem with a self-sorting recycling system, SortBot, that includes a sorting machine on site and a complementing mobile application that rewards recycling transactions. Technical aspect of SortBot was refined both in terms of software and hardware to have a intelligent image recognition system and a robust rotating mechanism that identifies objects and places them into correct compartments.

2. Introduction

This project addresses the large scope for improvement in the process of recycling both for the consumers as well as waste management authorities and companies. For the purpose of this project, the main target has been narrowed to focus on recycling in campuses. As discussed in the interim report, current methods employed in waste management include collecting all recyclables in the "Dry Mixed Recycling" bins and sorting them in the "Material Sorting Facilities". The problems associated with this are mainly dissuasion from recycling especially among students and contamination of recyclables.

1) Low rates of recycling among students

Due to budget cuts to authorities in charge of recycling and people still being confused about how to sort materials 'recycling rates have fallen in 2016 for the first time' (The Guardian, 2016).

As students are half as likely to recycle compared to anyone else in the UK (SITA UK, 2016) the project aims to give incentives to students to recycle more as primary research showed that 50% of Imperial students do not recycle on regular basis[1]. This is achieved by rewarding students every time they recycle achieving a social and environmental "win-win" system as 3 in 4 Imperial students said they would recycle more if there was a rewarding system[2]. Students will receive points they can later use for on campus purchases whereas the recycling process will be made easier and more economic by making the material recovery facilities redundant as the sorting of the materials will done on site.

2) Contamination of recyclables

The other problem that the project aims to overcome is the issue of contamination of recyclables as the 'rejected recyclable waste has (increased by) 84% in England since 2011' (BBC, 2016). The effects of this are adverse as a single non-recyclable in a bin can result in the whole recycling bin contents being rejected due to contamination. The project tackles this problem by automatically and accurately sorting items into the correct material category.

Therefore, the project would aim to create a feasible and sustainable solution that encourages recycling among students, increases the amount recycling without contamination and make the waste sorting process more efficient.

[1] Obtained from the questionnaire carried out among students, for full Questionnaire see Appendix 1
[2] Obtained from the questionnaire carried out among students, for full Questionnaire see Appendix 1

3. Design Criteria

3.1 Problem Research

In order to better understand the requirements of the problem, background research was done by contacting Imperial Estate Facilities Customer Service Centre. The information obtained is that the self-sorting items would be highly beneficial as this would accelerate the process of sending the materials to recycling by removing the large-scale sorting stage done in material recovery facilities. This would cut costs and eliminate manpower needed as most of sorting is done manually in these facilities making the whole process more economic and less wasteful.

3.2 Design Specifications

The design criteria have been kept very similar to the model outlined earlier in the interim report. The machine divides four categories of items (plastic, paper/carton, cans and general) and will be placed indoors being able to store one day's worth recyclable storage.

The table below outlines the design criteria already set in the Interim Report. The words in bold specify the criteria that have changed since the Interim Report and the rationale for these changes are specified below the following table.

Category	Criteria	Requirement
Manufacturing	Size	Has 4 standard sized compartments of 220 l volume each - **reduced to 120l for transport purposes.**
	Material	Suitable for indoor operation only
	Cost	Each unit made within £100 - **estimated cost £120**
Performance	Speed	Able to complete each sorting within 30 seconds – **40s**
	Storage Capability	Able to store one day's worth of recyclables in campuses and be cleared daily
	Reliability	Able to sort 5 distinct items common in campuses into 4 categories - **increased number of items that can be sorted**
	Accuracy	Able to distinguish common items purchased in campus at least 80% of the time
Operational Requirements	Power Requirements	Low power consumption. Have standby (idle) mode to save power when not in use. - **4 AA batteries**
	Maintenance	Able to run smoothly with servicing done twice a year
	Service Life	Able to be in use for a minimum of 5 years
Safety Requirements	Misuse/Abuse	Tray to place recyclables closes after every interaction preventing access into the compartments - **not available for presentation**

The capacity of the bins was reduced to half from 240L to 120L for transportation purposes during the demonstration. The estimated cost has increased to £120 for higher reliability and all of the costs are outlined in Section 5 of this report.

The overall time of operation has increased and each operation will take a total of 40 seconds: 15 seconds to take the picture and process the image, 5 seconds for the tray to go to the correct position and come back, 10 seconds to tilt, 5 seconds of waiting and 10 seconds to scan the QR code.

Regarding the power requirements, the motors will be supplied by 4 AA batteries and connected and disconnected by a switch instead of standby (idle) power saving mode. Finally, due to the improved image recognition technique, the machine will now be able to sort more than 5 items for the demonstration.

4. Design Consideration and Implementation

4.1 Design Overview

For the proper functioning of the self-sorting recycling machine, a combination of software and hardware is required to correctly classify four different categories: plastic, paper/carton, cans and general waste. The software part consists of a mobile app and the image recognition software. The app is available for both Android and iOS systems enables the users to interact with the machine and collect points. The hardware section is represented by the automated machine itself, in which the item to be sorted is placed. Each item is classified into the correct material category by using image recognition and is then sorted to the correct bin using a rotating tray operated by two servo motors. The automated sorting means that the waste will ready to immediately enter the recycling stage once it is collected from the machine.

4.2 Rotating Mechanism

In order to fulfil the design criteria and user needs the design that was devised is the following: four bins are placed in a circular shape for each of the four item categories (plastic/carton, cans, paper and general) looking similar to the picture shown below with three categories.

The sorting tray is centred above the four bins on a rod and is rotated around this centre so that it can face each of the four bins as necessary. Once the tray reaches the desired position it is tilted and the item to be sorted is dropped into the correct bin.

This structure will be covered by an exterior in order to protect the sorting tray and prevent the users of having the access to the bins. An opening will be made where the items to be sorted are placed by users and a QR scanner will be placed next to it to provide user interaction with the machine.

Figure 1: Model of the orientation of bins

The whole sorting process can be split into three technical parts: construction of the mechanical motor mechanism to rotate and tilt the tray to drop the item into the correct category, developing image recognition software that identifies the item and building an app that enables user interaction with the machine. The following sections outline the technical analysis, implementation and rationale for decisions for each of the three parts.

4.3 Motor Configuration

4.3.1 Servo Motor

Servo motor is a motor that allows the precise control of the position and angle of rotation. The inside of a servo motor consists of a DC motor, potentiometer and a control circuit. When the motor rotates the resistance of the potentiometer changes which allows the control circuit to determine the current position relative to the rest position, how much movement there is and in which direction (Reed, 2015).

The speed of rotation of a servo motor depends on its position as it is proportional to the difference between its current and desired position. Therefore, when the motor is far away from the desired position it will rotate fast and as it approaches the desired position it will slow down.

The motor has three connections: ground, power and a signal wire. It is controlled by sending an input which determines the angle of rotation through the signal wire. For example, if the input is given to be 90° the motor will rotate by this amount and then stop. However, a servo motor can usually only turn by 90° in either direction making the total angle of rotation 180° which when rotation of 360° is needed is a big disadvantage of this type of motor.

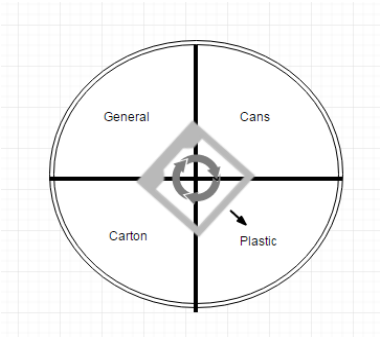
4.3.2 Servo Motor with Continuous Rotation

This is a special type of servo motor that does not have a limit on the range of motion and is able to rotate 360° without any restrictions (Carnegie Mellon, 2015). Furthermore, a servo motor with continuous rotation does not have the feedback control that allows for the position of the motor to be determined. Instead the motor is always defined to be at zero degrees. Because of this, the input signal no longer determines which position the motor should go to as it rotates indefinitely until it is stopped, but instead determines the speed of the rotation (Pololu, 2011).

For example, the input of 45 means the motor will rotate slower than for the input of 60 because the speed is determined by the difference between the desired and current position and in servo motor with continuous rotation the current position is always seen to be at zero. This is an advantage of this type of the motor as the speed of rotation can be easily controlled. However, its position can no longer be controlled and the motor rotates continuously once it is given an input.

4.3.3 Rotation of the tray

For the rotation of the tray, the servo motor with continuous rotation is used. This is because the normal servo motor does not have the flexibility to rotate 360° which is important for our design. The rest position of the tray is defined to be above the plastic bin as shown in the diagram below.



The tray only needs to rotate by either 90° left or right from the rest position to reach the paper/carton and can bins and by 180° for the general bin. As the motor will rotate continuously once the input is provided the time needed to rotate by 90° or 180° was measured manually and is found to be 2.15s and 4.30s respectively in the clockwise direction and 2.6s and 5.2s in the anticlockwise direction.

Figure 2: Top view of the tray above bins

In order to program the motor so that it can stop and operate correctly, once the time needed for the motor to reach the desired position has elapsed the motor is given an instruction to stop rotating. Following this the tray is tilted and motor is given an instruction to return to the original position in the same way just in opposite direction.

4.3.4 Tilting of the tray

As the tray is fixed on a rod with the servo motor with continuous rotation placed inside of it as shown in the pictures below.

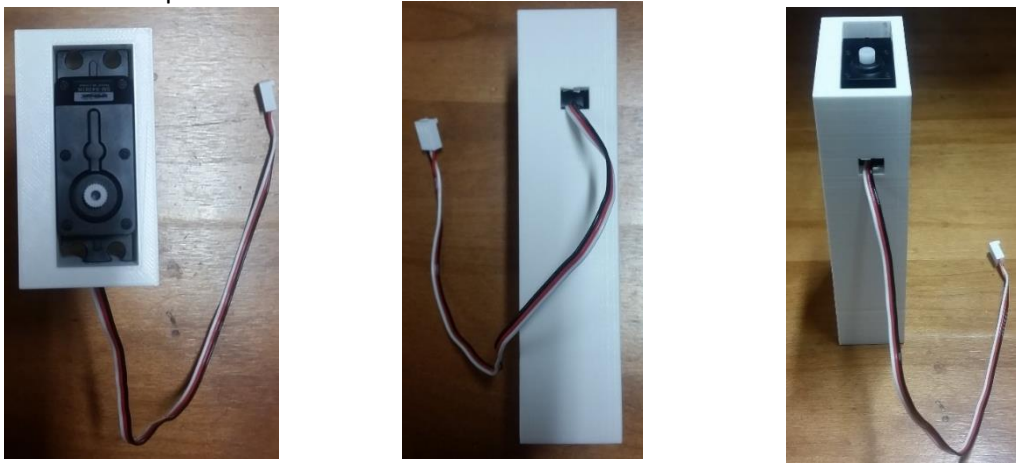


Figure 3: Picture of the 3D printed rod with the servo motor placed inside shown in Top view, Front view and Side view

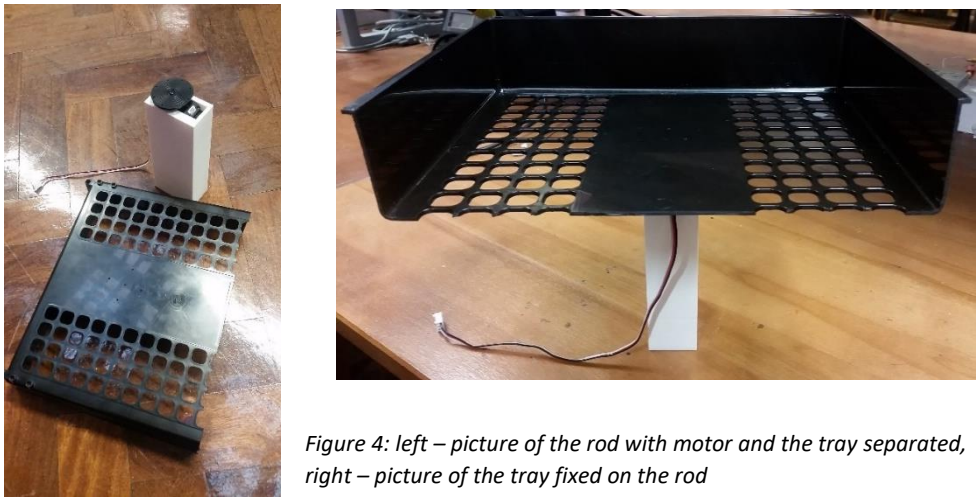


Figure 4: left – picture of the rod with motor and the tray separated, right – picture of the tray fixed on the rod

The servo motor that tilts the tray could not be fixed directly to the tray as this would prevent it from rotating. A solution that was devised was to break the rod that holds the tray into two pieces making the top piece holding the tray tilt and leaving the bottom one fixed as shown in the picture below. The two pieces are joined together with a servo motor which tilts the top rod along with the tray.

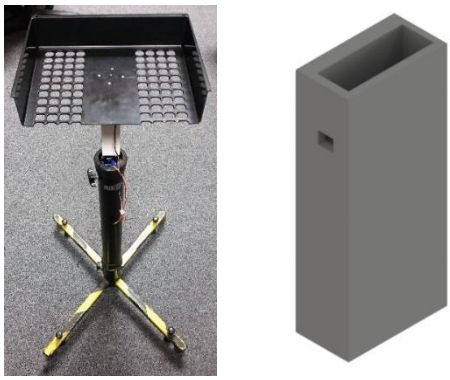
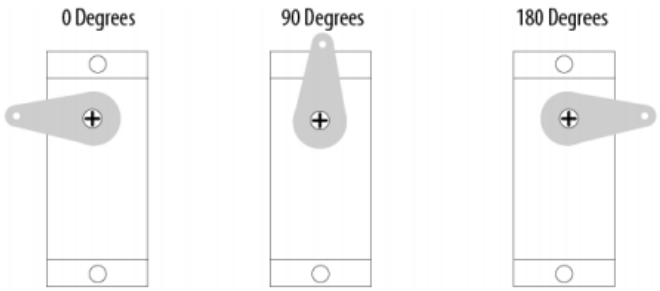


Figure 5: Left – picture of the tray with two rods with all the motors; Right – the top rod that was created in Autodesk and 3D printed

To implement the tilting, a normal servo motor is chosen as there is no need for 360° angle of rotation as the rod holding the tray only needs to tilt by 30° - 40°. As this angle is not big, the speed at which the tilting is done will not be too fast as the speed is dependent on the distance between the current and desired position as explained earlier. Furthermore, the input to the servo motor is the angle by which the rod needs to tilt which makes controlling of this type of motor easier. The part of the code that tilts the tray is shown below[3]:

```
void tilt() {
  myservo.write(120);          // turn the motor clockwise, input 120 defines the 30° tilting angle
  delay(2000);                 // wait in this position for 2s
  myservo.write(90);           // return the tray to the original position, input 90 defines the 30°
                               tilting angle in the anticlockwise direction
}
```



The initial rest position of the motor is 90°. Giving any value between 90° and 180° as input will turn the motor in the clockwise direction and for any value between 0° and 90° the motor will turn in the anticlockwise direction.

Figure 6: Servo motor position angles (Future Electronics, 2016)

[3] See Appendix 2 for the full code

4.3.5 Circuit Diagram

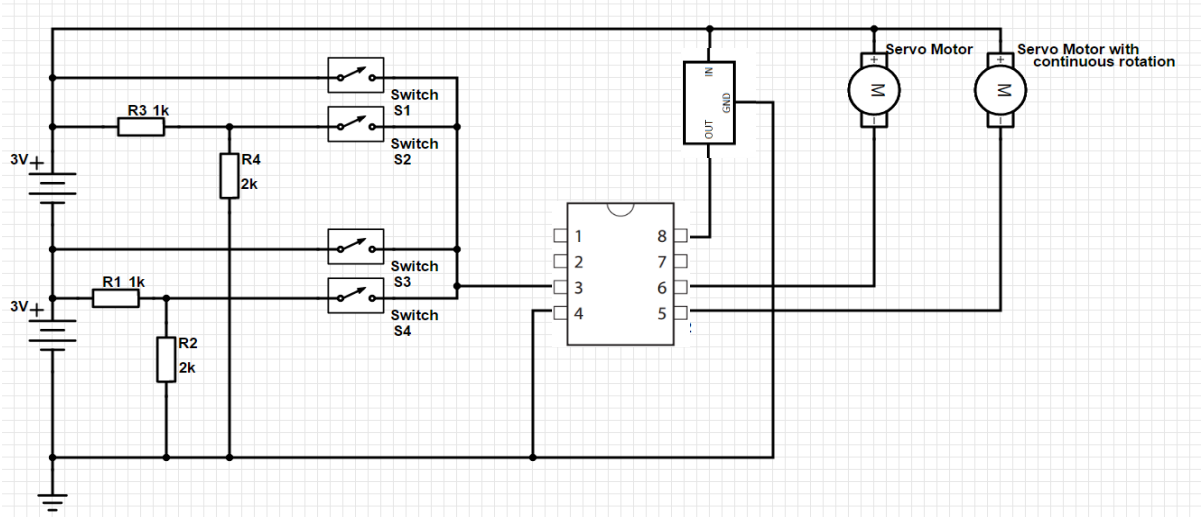


Figure 7: Circuit Diagram, first version

Potential Divider and Switches Configuration

Ideally, the output of the image recognition software should be directly inputted to the AtTiny85. However, as this is out of the scope of the project, once the software identifies the category of the item to be sorted this information is fed into the circuit manually using switches.

In order to do this a mapping between the category type and different voltage values is produced. As the power supply for the circuit are four 1.5V batteries connected in series with a total voltage of 6V the mapping that is done for the four different categories is as follows:

1. Plastic → 1.5V
2. Paper/Carton → 3V
3. Cans → 4.5V
4. General → 6V

This was first implemented by using potential dividers as shown in the circuit diagram above in order to obtain the desired voltage values from the 6V supply. However, after testing this model a short circuit was detected when turning the switches on and off. This was due to the direct connection from the positive voltage supply of the battery to GND when the switch is closed which led to the malfunctioning of the circuit.

Hence a new circuit was designed following a wired remote control circuitry patterns. Here, four resistors in parallel with four switches are connected in series forming a potential divider with a single 5kΩ resistor as shown in the circuit diagram below. This configuration gives 4 different input voltages controlled by switches with no risk of creating a short-circuit.

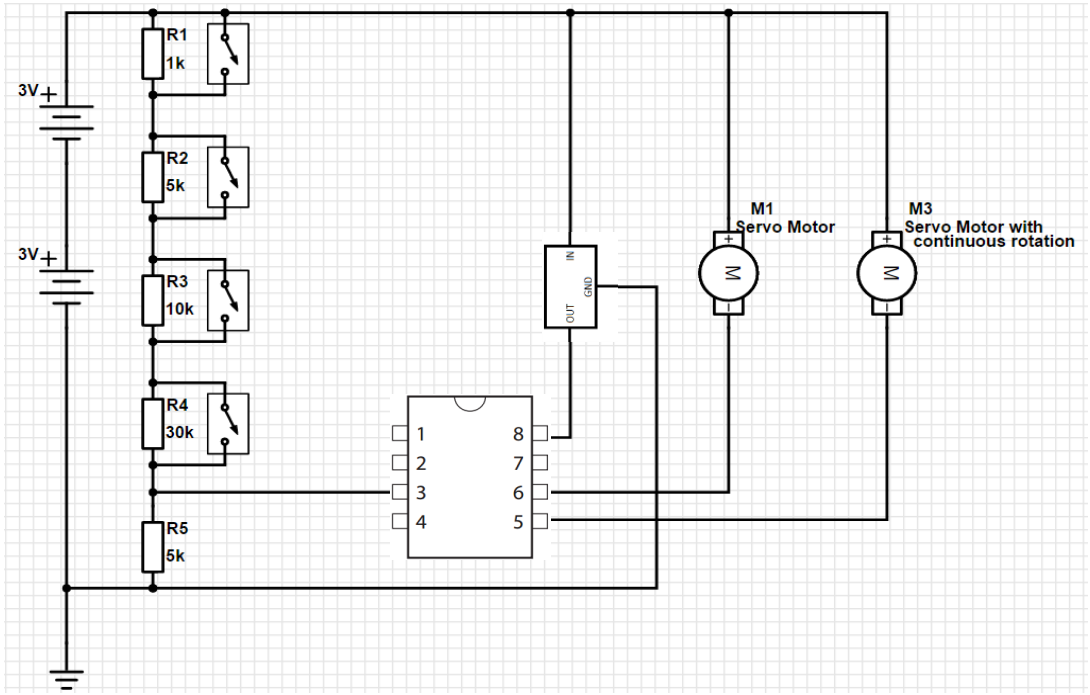


Figure 8: Circuit Diagram, final version

Another benefit of this circuit is a possibility to reduce the number of switches from 4 to 2 following this Boolean diagram as either switch can be on or off giving four possible voltage values:

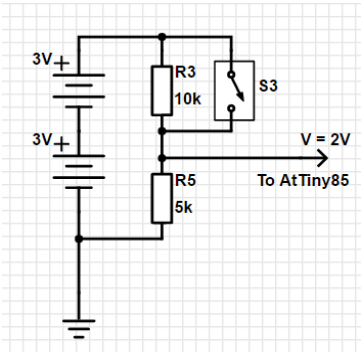
Switch A	off	off	on	on
Switch B	off	on	off	on
V_{out}	min ~ 1 V	~ 2V	~ 4V	max ~5V

However, for testing and demonstration purposes the four switches will be kept to represent more clearly each item category: plastic, cans, paper/carton or general.

As the circuit will be controlled manually during the demonstration, when there is no item to be sorted all of the switches should be closed. The resistor values are chosen to give the following voltage values to the AtTiny85 depending on the item category:

1. Plastic → S4 is open → 1V
2. Paper/Carton → S3 is open → 2V
3. Cans → S2 is open → 3V
4. General → S1 is open → 5V

Once the category of the item is obtained as the output from the image recognition, the appropriate switch is opened so that the potential divider is created with R5 and voltage value representing the specific category is sent to the input of AtTiny85.



For example, it the item is detected to be carton the left half of the circuit will have the configuration shown on the left. The switch S3 is open so that resistor R3 forms a potential divider with R5 creating a 2V signal that is sent to the input of the AtTiny85. As all other switches are closed all other resistors are shorted out so they do not affect the circuit in any way.

Figure 9: Left half of the circuit for carton input

AtTiny85 Microcontroller Operation

The AtTiny85 receives voltage value as analogue input which represents a specific item category. Since the potential dividers are used to generate four different input values and because the batteries will not be able to produce 6V due to usage over time, a small percentage of inaccuracy is expected. In order to handle this the code will deal with the issue as follows:

```
if ((voltage > 1.5) && (voltage < 2.5)) {          // anything within the 2V range is classified as carton
        carton();
}
```

On the output side, pins 5 and 6 of the AtTiny85 are used to control the servo motor with continuous rotation and the normal servo motor respectively. AtTiny85 will process the input voltage and depending on its value will send signals to the two motors in order to rotate the tray by a specific amount, tilt it and finally return it to the original position.

Firstly, the motor with continuous rotation is energised and rotated so that it faces the correct bin. During this time the servo motor used for tilting of tray is at rest. Once correct position is reached the rotating motor stops and the control is passed to the motor which tilts the tray so that the item is dropped into the correct bin. When this action is completed the motor which tilts the tray is turned off and the motor that rotates the tray is activated to return the tray to the starting position.

For example, if the image recognition software identifies the item as carton the appropriate switch is closed manually. AtTiny85 receives the value of 2V as analogue input and makes the tray rotate 90 degrees so that the tray faces the bin containing paper/carton items. A fraction of the code that shows how the motor is controlled when the item to be sorted is carton is shown below[4]:

```
void carton() {
  myservo.write(88.5);          // turn the motor clockwise, input 88.5 defines the speed of rotation
  delay(2150);                  // keep rotating for 2.15s until the carton bin is reached
  myservo.write(91);            // stop the motor at this position, input 91 defines zero motion
  delay(5000);                  // for 5 sec here until the tray is tilted to drop the item
  myservo.write(99);            // turn anticlockwise to original position, input 99 defines the speed
  delay(2600);                  // keep rotating for 2.6s until the starting position is reached
  myservo.write(91);            // stop the motor at the starting position and wait for next action
}
```

[4] See Appendix 2 for the full code

4.4 Image Recognition

4.4.1 Image Recognition Design Selection

The project requires a select set of common recyclables to be classified into the four categories through instance-level recognition (Vedaldi & Zisserman, 2014). In the interim report, several methods of identification were discussed to distinguish objects placed in the tray through photographs taken. Since they can be placed in any direction by the students, a method that is immune to the object orientation is crucial. Moreover, image recognition programs also require implementation of machine learning that increases accuracy by learning from previous results. Methods evaluated are as follows:

1) Scale-Invariant Feature Transform (SIFT)

SIFT is an algorithm that is invariant to rotations, translations and scaling transformations of images (Scholarpedia, 2012). Therefore, the item can be recognised even if there are differences in position or lighting (Vedaldi & Zisserman, 2014).

In order to better understand this process, Mr Juil Sock, a PhD student specialising in image processing was consulted with the recommendation of the group's supervisor. It was discussed that a database would be created with images of each object from different angles with a common label, along with information on their SIFT features.

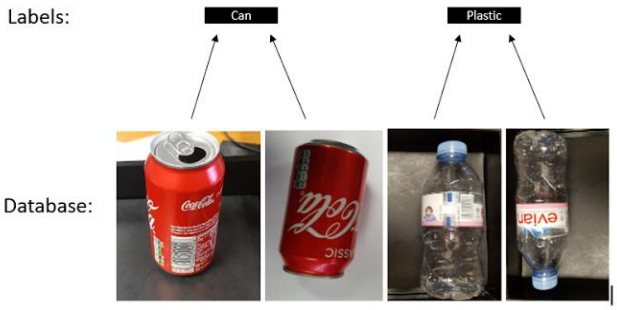


Figure 10: Images of objects in different orientations

As seen in the figure above, images of objects in different orientations and lightings can be taken and made to be associated with the same label created such as water bottles being linked to "Plastic". When a new image of interest is inputted, it is compared against the existing database to retrieve the final label. By using machine learning, the algorithm can learn from the data and results input before to improve the performance of the following results. As the database increases, the accuracy of the system is also improved enabling many different objects to be placed from various angles.

2) Google Cloud Vision API

Google Cloud Vision API is a computer vision API from the Google Cloud Platform that can be integrated in websites and applications to obtain information on the content of an image. The Vision API can detect faces, logos, landmarks, image attributes etc. Vision API is very powerful and has plenty of functions - the only function of interest to this project is Label Detection. Labels are the objects/categories that the API can identify in the image.

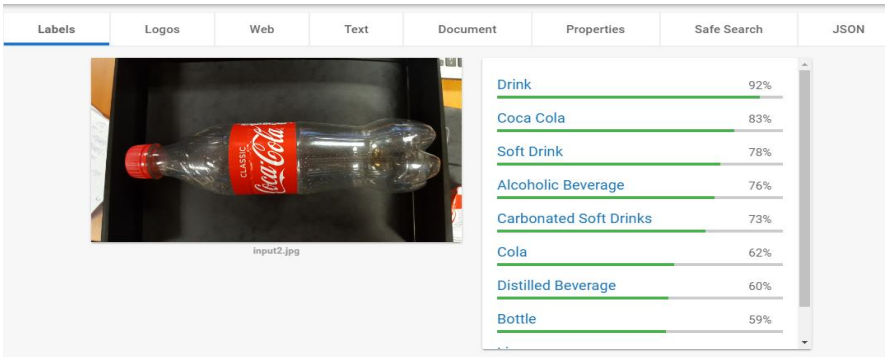


Figure 11: Results from Google Cloud Vision Interface

The figure above depicts the label created when an image is fed into the Google demonstration interface that aims to only illustrate how the API works. To use this function, the suitable Vision API must be added into the code programmed by the user in Python or other programming languages. The programme then imports the Vision library to obtain a list of labels associated with the input image. Deep machine learning through the enormous Google database allows for reliable labelling. The API also scores the level of certainty for each label (Google-Cloud, 2014). While the input image to the Vision API can be anything, for the use in this project it is a single object against a plain black background.

Design Selection

SIFT algorithm focuses on areas of interest with intricate details such as designs on building to compare between images. Thus, it works more accurately for images of 'high texture' i.e. those with more variety in colours, designs and details. However, many types of recyclables may not have such distinguishable features making SIFT more ineffective.

On the other hand, the Google Cloud Vision API has a higher level of accuracy compared to the SIFT algorithm that are written from scratch from the group as it harnessed the Google's existing large database. One of the main aims of SortBot is to prevent the contamination of the recyclables, which is why its image recognition software would be better implemented by importing the Google Cloud Vision API. Google Cloud Vision API was also chosen over other computer vision APIs for its ease of use, the availability of a good open source code and adaptability to different programming environments.

4.4.2 Image Recognition Concept Development

Test Procedure

To formulate a suitable algorithm for the image detection process, the Vision API had to be tested with the target objects. A list of common items that are recycled by the students in the Imperial College London main campus was obtained through a campus-wide survey.

Images of these items were taken against a black background, keeping other elements such as angle and lighting constant to simulate the photographs that would be taken internally by SortBot. These were the inputs to the image recognition programme to obtain the labels and their corresponding scores.

Test Results

No.	Test object	Label	Score	Category
1	Paper	handwriting	0.8206787705	Paper
		art	0.666133523	
		document	0.6128236055	
		writing	0.610511899	
2	Newspaper	text	0.940867424	Paper
		advertising	0.6218293309	
		document	0.5373576283	
3	Water bottle	drink	0.8869991899	Plastic
		alcoholic beverage	0.7854443789	
		distilled beverage	0.676841557	
		bottle	0.615100503	
4	Innocent juice bottle	drink	0.7516778708	Plastic
		bottle	0.6988299489	
5	Coca-Cola can	red	0.9574187398	Can
		coca cola	0.946059525	
		drink	0.9212352037	
		carbonated soft drinks	0.8557853699	
6	Energy drink can	drink	0.7943568826	Can
		energy drink	0.7339280248	
7	Dirty paper plate	white	0.9376140237	Contaminated
		porcelain	0.7050687075	
		ceramic	0.6711545587	
		food	0.6440527439	

The data obtained above would later be interpreted and analysed to formulate the algorithm that sorts specifically chosen labels to recycling categories. All other labels that do not fit in any of the categories are ignored. The algorithm is first refined to sort only these select items correctly and reliably and the database of items to be sorted can be expanded in the future.

4.4.3 Image Recognition Design Challenges

1) Contamination

One of the key problems with current recycling methods was identified as contamination; a single product with food, for example, can result in the disposal of the entire collection of recyclables. Thus, it is crucial that SortBot does not include such objects into the recyclable categories incorrectly. A ‘general’ category is present that will encapsulate all objects that do not contain the identified keywords.

Moreover, an additional checking element is added into the algorithm. If the Vision API identifies words associated with contamination such as ‘food’, the object of interest will be automatically sorted into ‘general’. Thus paper plates, even though recyclable themselves, are stained with food and can be sieved out.

The opportunity cost of implementing this checking process is that certain uncontaminated objects will be wrongly sorted into ‘general’. For example, a clean, unused paper plate could still have that label due to its association with ‘food’. However, the cost of contamination outweigh the cost of misplacing some of the recyclables. Therefore, this checking element is also implemented ensuring the machine is even more competent in solving the problem identified.

2) Overlapping labels
A generic label detected by the Vision API can correspond to multiple categories. For example, the label 'coca cola' can correspond to both the soft drink bottle and the can, resulting in an overlap of categories.

To construct a more reliable system, such brand names and generic labels have to be ignored. Even if an unsuitable generic keyword (e.g. coca cola) is one of the labels, the programme will iterate through the list of labels provided by the Vision API to find the specific ones (e.g. bottle, plastic) that SortBot is able to categorise with confidence - these categories are then stored in another list.

A further problem that could occur is when the programme detects labels that are sorted into conflicting categories. For example, a plastic bottle is labelled as ‘bottle’, ‘plastic’ and ‘glass’. The resulting list of categories is ‘plastic’, ‘plastic’ and ‘glass’ (note that SortBot currently does not include a bin for glass, so actual glass items should go into ‘general’). The algorithm is refined to find the category that occurs the most frequently to solve this problem, further strengthening the need to store the possible categories in a list.

3) Inability to identify some items
As a direct cost to using the solution in 6.3.3.b, the Vision API has been shown to frequently failing in identifying some items in a way that SortBot can categorise them meaningfully.

The most inaccurate labels the Vision API has given to a Coca-Cola can are ‘art’, ‘wheel’, ‘tire’ or just ‘red’. At best, the API is able to label it with ‘coca cola’, ‘soft drink’, ‘carbonated soft drinks’ and ‘drink’. As stated in 6.3.3.b, such generic labels and brand names must be ignored since it can refer to both the plastic bottle and the can.

SortBot will instead implement a conditional statement to properly sort these cans. When the programme detects that an item is labelled as ‘soft drink’, it will check if it is also labelled as ‘bottle’. The presence of the label ‘bottle’ implies that the item is a beverage bottle, while its absence implies that it is a canned drink. In summary, for SortBot to sort cans, it analyses generic labels such as ‘soft drink’ but on stricter conditions by checking the rest of the list of labels too.

```
if ('soft drink' in listOfLabels):
        if ('bottle' not in listOfLabels):
                sort item in 'can'
        else if ('bottle' in listOfLabels):
                sort item in 'plastic'
```

4.4.4 Program Implementation

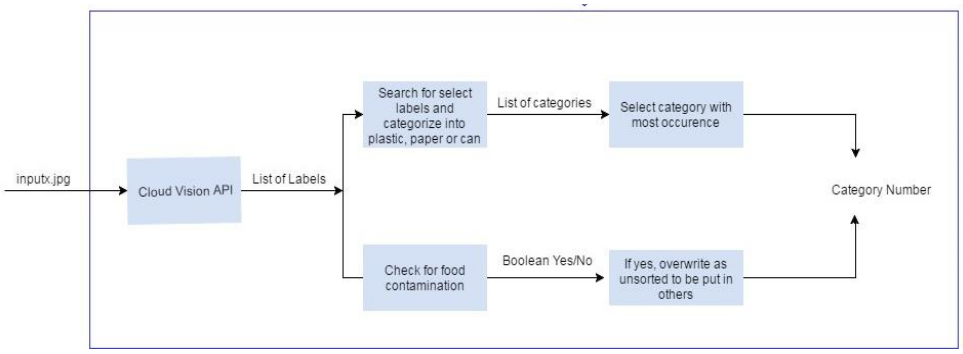


Figure 12: Illustration of the program implementation

The overall programme implementation is illustrated in the figure above showing the image being input and the final category number is obtained as the output. The data obtained from testing and experimentation earlier was used to create a table of labels to be associated with the categories as seen below:

Category	Plastic	Paper	Can	Contamination
Labels	Plastic Bottle	Paper White Writing Document Calligraphy Poster Book	Can Drink (on the condition that the list does not include 'bottle' energy drink)	Food Coffee Plate Dishware Tableware Saucer Cup

Based on this, the complete programme is written in Python and is built upon an open source code available (Google Cloud, 2016). It accepts input images named in the format 'inputx.jpg' (where x is an integer) and stored in the folder named 'resources' in the same directory of the programme. To illustrate the entire structure of the programme written, the following pseudo code has been included:

```
# Iterating through the entire list of labels
for label in listOfLabels:
        # Checking for contamination is the utmost priority
        if "food" in label:
                add "contaminated" to listOfCategories
                containsFood = True
        else if "coffee" in label:
                add "contaminated" to listOfCategories
                containsFood = True

        else if "paper" in label:
                add "paper" to listOfCategories
        else if "plastic" in label:
                add "plastic" to listOfCategories
        else if "can" in label:
                add "can" to listOfCategories

        else if "bottle" in label:
                add "plastic" to listOfCategories
        else if "white" in label:
                add "paper" to listOfCategories
```

As seen on the left, the code iterates through the list to find each of the labels stated In Table X, and creates a new list placing them into their respective categories. At the same time, it is also checking for labels that indicate a possibility of the item being contaminated by food by using a Boolean variable. If this is true, it takes priority regardless of whether labels indicating recyclables are also found or not and this is automatically sorted into 'General'. Otherwise, for items that can be recycled, the programme will find the category with the most occurrences by importing the statistics library and applying the mode function - this is the category that the item will be physically sent to.

Category	Category number
General	0
Paper	1
Plastic	2
Can	3

Some items, such as non-recyclables, will not be categorised at all. The programme will check for such situations and move on the next part of the code if needed. The programme labels all contaminated items and uncategorised items as 'general'. Each category is assigned a number, which is the information being sent to the motor and the accompanying mobile app according to the table on the left.

4.5 The Mobile App

4.5.1 Rationale for Decisions

The app directly connects the user and the machine as it takes the information from the image recognition as input and converts it into points for the user that can be redeemed. Each user account will have a personal QR code that can be scanned after every operation to accumulate points. The SortBot app will allow the user to keep track of the points achieved as well as the items recycled.

The app will be built in iOS environment instead of Android and hence, the app will be coded in Swift instead of Java. In the first instance, the platform of Ruby on rails was considered due to its compatibility with both Android and iOS and easy access to database. This would however consist of a web app with limited functionalities which design is shown below:

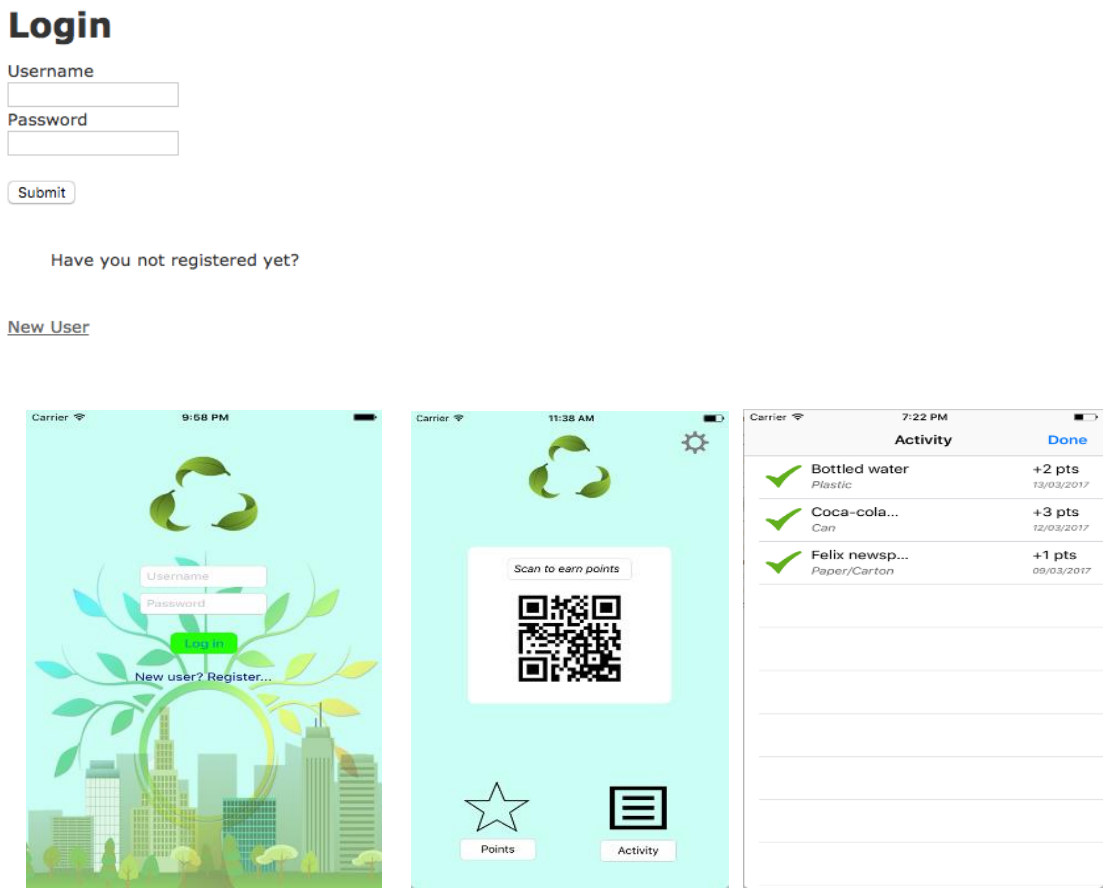


Figure 13: Top – Web App clearly showing limitations, Bottom – Screenshot from the iPhone simulator in Xcode

Hence, the Xcode option was selected to create a proper mobile app with a wider design option as shown above. After logging into the user account, the personal QR code will appear ready to be scanned in the SortBot machine compartment:

The main functionality of the app is providing each user with a unique QR code, which the user scans before recycling each item and receives points based on the material of the recycled item. Therefore, the main screen for a logged-in user consists of his or her QR code and links to the page displaying their points and their activity. The latter consists of all their previous transactions, displaying the item recycled, its material, the number of points received and the date of the transaction. Due to time constraints, we have not considered the QR scanning technology in much detail. However, given the rapidly increasing popularity of it, we believe that this aspect of our project would not be overly difficult.
Ruby on rails will be used as a server, the back-end of the app that handles the persistent data of the mobile app, storing the users and its corresponding passwords.

Points Allocation

The allocation of points is based on the table below:

Number	Meaning	Points
1	Plastic	3
2	Paper	1
3	Cans	3
4	General	0

The app will include a section showing the sum of all points earned based of the items put in the machine and another section with the history of all transactions.

For demonstration purposes, instead of scanning the QR code, the user will be asked to submit one number 1, 2, 3 or 4 which makes it easier to connect the machine to the image recognition output. Later phases of the project development will have as an input a file contained in the QR code representing the item that has been inserted and its corresponding material. To sum up, the SortBot app will read the input file from the QR code which is already categorised by the image recognition function in the following way:

- Bottled water - plastic 2 points
- Coca-Cola can - can 3 points
- Felix newspaper - carton/paper 1 point

4.6 Rationale for Decisions

The overall function of SortBot was split into three technical sections: developing the image recognition software, constructing the rotating and tilting mechanism by controlling the motors and building the app. By connecting all of the technical sections together a fully working system can be obtained. Given the limited amount of time, automating all of the processes is out of the scope of this project. Instead, each technical part will be fully operating on its own and output of each stage will be fed into the next one manually.

5. Industrial Design and Manufacturing Considerations

In order to estimate the cost of designing the final product all of the components that were bought so far and their costs are included in the table below:

Component	Quantity	Price per Unit (£)
Arduino T010051 Digital Continuous Servo Module (360°)	2	11.25
Miniature 240V SPDT Vertical PCB Slid	4	0.39
Battery AA Alkaline 1.5V (Energiser)	4	0.24
RVFM Foolscap Letter Tray - Black	1	2.07
ST L7805CV +5V 1A Voltage Regulator	1	0.28
Modelcraft RS 2 JR BMS-410C Plastic Gear JR Standard Analogue Servo	1	5.12
Adafruit 1967 Mini Pan-Tilt Servo Controlled Brackets	1	16.18
Atmel ATTINY85-20PU, 8bit AVR Microcontroller, 20MHz, 8 kB, 512 B Flash, 8-Pin PDIP	1	1.64
3D printed rod of dimensions 5x10x15 120g	1	3.60
	Total Cost:	**53.91**

This only includes the cost of building the mechanical system of the machine which includes motors for rotation and tilting, rods for the support and mounting and the sorting tray. The list above also includes the cost of building the circuit with all the components that control the operation of the motors.

The items that still need to be ordered for the machine to be complete are the QR scanner that enables the user to interact with the machine through the app and a camera that needs to be placed above the tray in order to take images of the items to be sorted. After looking through the possible options the price of these items are as follows:

- QR scanner - £10.99, price taken from (Amazon, 2016)
- Camera - £24, price taken from (eBay, 2016)

To build and manufacture a final machine further costs need to be considered such as constructing the cover for the exterior of the machine that protects the sorting mechanism as well as the four bin compartments. The estimated cost for this is around £30 as the materials needed to do this are not expensive.

At the current stage of the project, the Vision API is being used at a rate of roughly 50 requests per day, for one week - corresponding to an estimation of 350 requests. The API is free to use up until 1000 requests per month (Google Cloud, 2014). When SortBots are ready to be deployed to university campuses, it would cost £1.50 per month to continue using the API beyond 1000 requests per month. Finally, summing up all the costs of manufacturing the overall estimated cost for the entire machine is £120.

Figure 14: 3D Model of the entire SortBot Machine

6. Project Management

The work was equally split between the seven group members. Regular meetings were held both to keep track of the progress and to discuss design choices and new ideas.

The project was split into four main areas so that each student could give his/her own contribution to the development of the final design, according to one's interests and skills. The individual allocations are shown in the table below.

Task	Group Members
Image Recognition	Abhinaya Mathivanan Sze Tyng Lee
Sorting Mechanism	Wendy Lu Chen Jacopo Carrani Katarina Boskovic
App Design	Octavian Rosu
Website Design	Guo Liang Liew

The Gantt chart shown below was constructed in order to have an estimate on how much time should have been dedicated to the different stages of the project and to always be aware of the deadlines the group had to meet.

				January			February				March		
	Start	End	Duration	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3	Week 4	Week 1	Week 2	Week 3
Assign responsibilities	09.01.2017.	09.01.2017.	1 day										
Agree on work allocation	09.01.2017	09.01.2017.	1 day										
Research and discuss different design options	09.01.2018	23.01.2017.	2 weeks										
Agree on one option to implement	23.01.2017.	23.01.2017.	1 day										
Split the writing tasks	16.01.2017.	16.01.2017.	1 day										
Design and construct the step motor	27.01.2017.	28.02.2017.	1 month										
Design the app	27.01.2017.	28.02.2017.	1 month										
Work on nad implement image recognition	20.01.2017.	28.02.2017.	1 month										
Order components	10.02.2017.	10.02.2017.	1 day										
Submit the interim report	08.02.2017.	08.02.2017.	1 day										
Build a prototype	13.02.2017.	13.03.2017.	1 month										
Create a website	13.02.2017.	13.03.2017.	1 month										
Submit the final report	13.03.2017.	13.03.2017.	1 day										
Presentation	21.03.2017.	21.03.2017.	1 day										

Figure 15: Gantt Chart

7. Future Work

The final aim of the project, which could not be achieved due to the time constraint, would have been to fully automate the sorting process by integrating the independent systems into one holistic machine. As of now, the output of the image recognition part must be manually fed to the sorting mechanism, but after further work and refinement, there would be no need for human interaction between stages. Also, we aim to incorporate SortBot into university campuses with an interactive user interface embedded and an improved aesthetic aspect.

Moreover, our ultimate goal is to create a self-sustainable system where the profits from selling the recyclables to recycling centres could be used for maintenance and upgrade of SortBot. Introducing a low power mode in the operating modes of SortBot can help save energy usage and thus reduce the cost of operation of SortBot. We will coordinate liaison with external businesses to build an attractive rewarding scheme. This could be achieved by partnering with a payment app like YOYO Wallet and businesses targeting students such as online retailers and food delivery companies.

The technology can also be improved to be able to detect multiple objects at the same time, and to accommodate more categories of recyclables. In the long term, the SortBot technology can be promoted as a possible approach for wider applications such as community councils and mobile waste trucks.

By harnessing more advanced image recognition technologies, this recycling method can be implemented to wider applications. For example, Google has recently announced of algorithms that detect multiple objects in a single image. With this, SortBot can be adapted to sort larger amount of waste in one go such as on waste management trucks (Collins, 2014).

8. Conclusion

This project has proposed a solution to improve the state of waste management in universities making recycling more appealing and effortless. The end goal is to increases recycling rates at campuses and reduce wastage of resources by automating the process completely. The technical solution designed is SortBot, a holistic system of a sorting machine and mobile application that rewards recycling.

After research and experimentation, SortBot was designed and made to have complementing software and hardware to accomplish the task. The image recognition aspect was produced with an external API and an algorithm that interprets the results obtained. The rotating mechanism was built completely with servo motors and 3D printed compartments that function reliably with appropriate coding. The independent parts work smoothly and can be integrated to form a full-functioning machine given more time. This technology can be improved and implemented to revolutionise the way recycling is done.

References

Amazon. (2016). *Amazon*. Retrieved March 12, 2017, from OneBird M3 2d Qr Wired Handheld USB Barcode Scanner Reader Support Mobile Payment Computer Screen Scanner with a Bottle Opener: https://www.amazon.co.uk/OneBird-M3-Handheld-Barcode-Computer/dp/B016EI7Z2W/ref=sr_1_55?ie=UTF8&qid=1489340836&sr=8-55&keywords=qr+barcode+scanner

BBC. (2016, August 23). *Rejected recyclable waste up 84% in England since 2011*. Retrieved from BBC: http://www.bbc.co.uk/news/uk-37159581

Carnegie Mellon. (2015). *Continuous Servo*. Retrieved March 03, 2017, from Carnegie Mellon Robotics Academy: http://www.education.rec.ri.cmu.edu/content/electronics/boe/robot_motion/1.html

Collins, K. (2014, September 08). *Google's Imaging Tech will Advance Robots and Research*. Retrieved March 09, 2017, from Wired Technology: http://www.wired.co.uk/article/google-research-object-recognition

eBay. (2016). *eBay*. Retrieved March 12, 2017, from HD 1080p Video Camera DV similar gopr or o SJ4000 Sports waterproof: http://www.ebay.co.uk/itm/HD-1080p-Video-Camera-DV-similar-gopr-or-o-SJ4000-Sports-waterproof/131896358605?_trksid=p2141725.c100338.m3726&_trkparms=aid%3D222007%26algo%3DSIC.MBE%26ao%3D1%26asc%3D20150313114020%26meid%3D24ff3ee149c84920810036f323ce7fd2%26p

Future Electronics. (2016). *Servo Motors Control and Arduino*. Retrieved March 10, 2017, from Future Electronics: http://www.inmoov.fr/wp-content/uploads/2015/02/Introduction-to-Servo-Motors-Arduino.pdf

Google Cloud. (2014). *Pricing*. Retrieved from Google Cloud Platform: https://cloud.google.com/vision/docs/pricing

Google Cloud. (2014). *Usage Limits*. Retrieved March 03, 2017, from Google Cloud Platform: https://cloud.google.com/vision/docs/limits

Google Cloud. (2016). *Google Cloud Vision API Python Samples*. Retrieved from Github: https://github.com/GoogleCloudPlatform/python-docs-samples/tree/master/vision/cloud-client

Google Could. (2014). *Authentication*. Retrieved March 03, 2017, from Google Cloud Platform: https://googlecloudplatform.github.io/google-cloud-python/stable/google-cloud-auth.html

Google-Cloud. (2014). *Using the Vision API*. Retrieved March 03, 2017, from Google Cloud Platform: (https://googlecloudplatform.github.io/google-cloud-python/stable/vision-usage.html

Pololu. (2011, July 26). *Continuous-rotation servos and multi-turn servos*. Retrieved March 03, 2017, from Pololu: https://www.pololu.com/blog/24/continuous-rotation-servos-and-multi-turn-servos

Reed, F. (2015). *How Servo Motors Work*. Retrieved March 03, 2017, from Jameco: http://www.jameco.com/jameco/workshop/howitworks/how-servo-motors-work.html

Scholarpedia. (2012). *Scale Invariant Feature Transform*. Retrieved from Scholarpedia: http://www.scholarpedia.org/article/Scale_Invariant_Feature_Transform

SITA UK. (2016, April 22). *University recycling survey gives insight into student recycling behaviours*. Retrieved from SITA UK: http://www.sita.co.uk/news-and-views/press-releases/university-recycling-survey-gives-insight-into

The Guardian. (2016, December 15). *Recycling rates in England drop for first time*. Retrieved from The Guardian: https://www.theguardian.com/environment/2016/dec/15/recycling-rates-england-drop-first-time

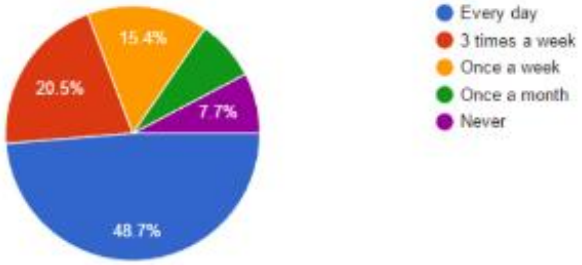
Vedaldi, A., & Zisserman, A. (2014). *Recognition of object instances practical*. Retrieved March 7, 2017, from Oxford Visual Geometry Group: http://www.robots.ox.ac.uk/~vgg/practicals/instance-recognition/index.html

Appendix

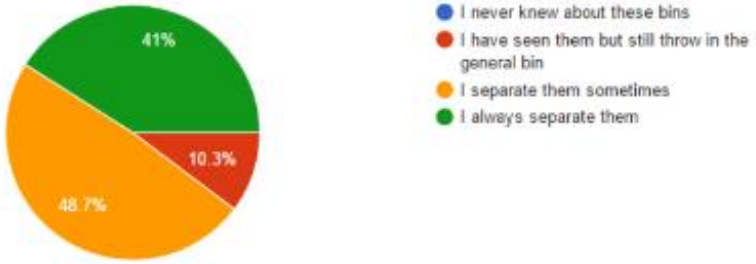
Appendix 1 – Results of the primary research carried out among students

Questions 1-3:

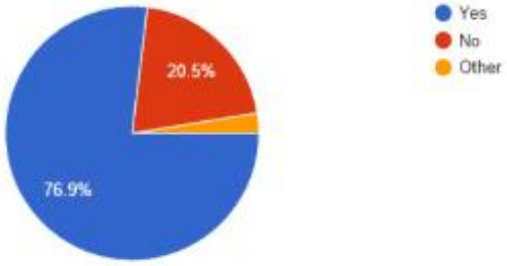
How often do you recycle on the campus? (39 responses)



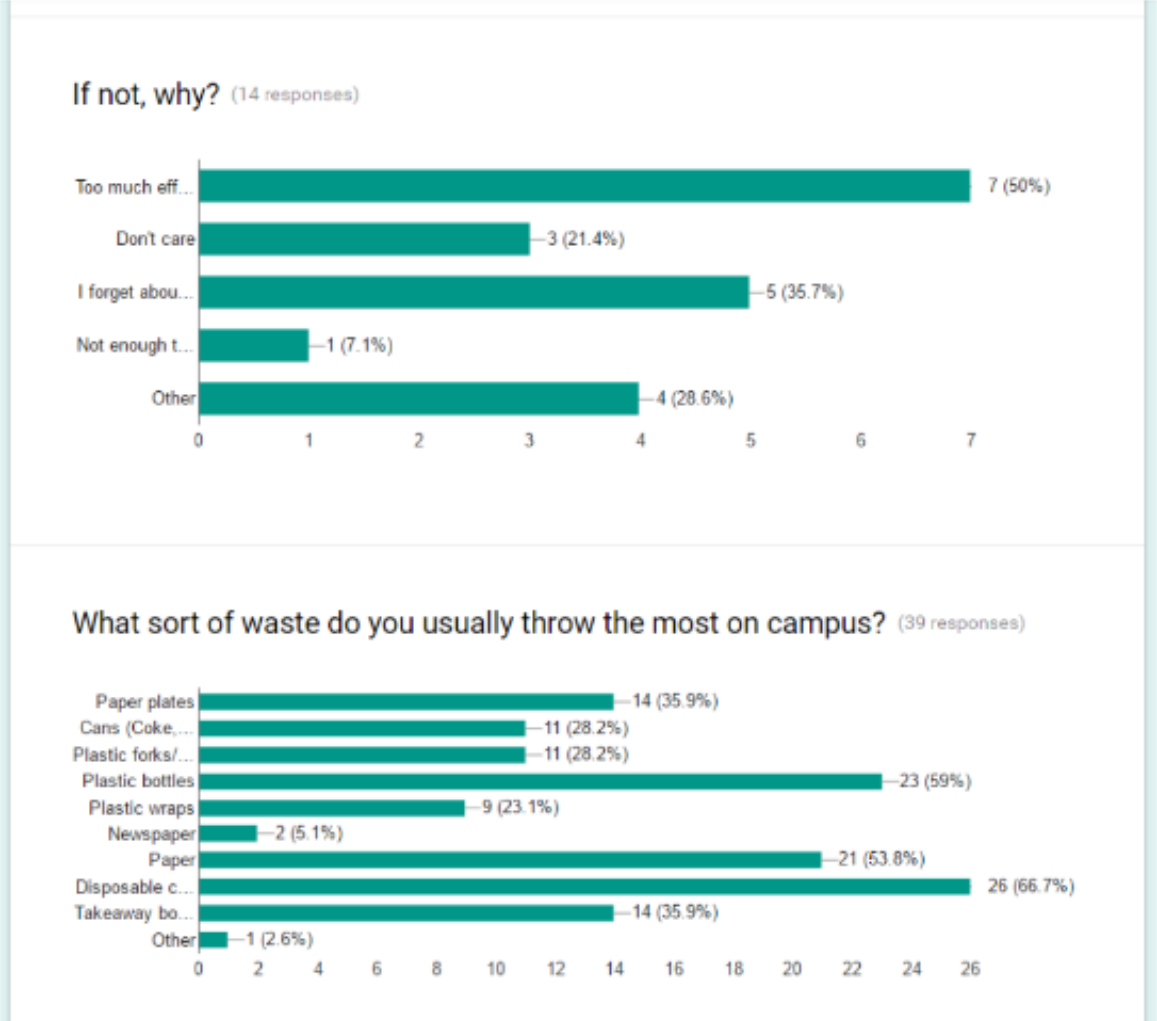
There are bins in campus for Paper (Blue), General (Black) and Dry Mixed (Green). Do you separate your trash?
(39 responses)



Would you recycle more if given points (like Yoyo) which can be redeemed?
(39 responses)



Questions 4-5:



Appendix 2 – Code for Controlling Motors

```
#include <Servo.h>

Servo myservo;  // create servo object to control a servo

const int AnalogueInput = 3, PWMContServo =  0, PWMTiltServo = 1;

void setup() {
  //myservo.attach(9);  // attaches the servo on pin 9 to the servo object
  pinMode(AnalogueInput, INPUT);
  pinMode(PWMContServo, OUTPUT);
}

void loop() {
    int voltage = map(AnalogueInput, 1, 6, 1, 1024);
    if ((voltage < 1.5)) {
      plastic();
    }
    else if ((voltage > 1.6) && (voltage < 2.5)) {
      carton();
    }
    else if ((voltage > 2.6) && (voltage < 3.9)) {
      can();
    }
    else if (voltage > 4) {
      general();
    }
}

void plastic() {
  analogWrite(PWMContServo, 91);
  delay(5000);
}

void carton() {
  analogWrite(PWMContServo, 88.5);
  delay(2150);
  analogWrite(PWMContServo, 91);
  delay(5000);
  analogWrite(PWMContServo, 99);
  delay(2600);
  analogWrite(PWMContServo, 91);
  delay(5000);
}

void can() {
  analogWrite(PWMContServo, 99);
  delay(2600);
  analogWrite(PWMContServo, 91);
  delay(5000);
  analogWrite(PWMContServo, 88.5);
  delay(2150);
  analogWrite(PWMContServo, 91);
  delay(5000);
}

void general() {
  analogWrite(PWMContServo, 88.5);
  delay(4300);
  analogWrite(PWMContServo, 91);
  delay(5000);
  analogWrite(PWMContServo, 88.5);
  delay(4300);
  analogWrite(PWMContServo, 91);
  delay(5000);
}
```

Appendix 3 – Image Recognition Code

```
def
sortbot():
                    import io
                    import os


                    # Imports the Google Cloud client library
                    from google.cloud import vision
                    from statistics import mode


                    # Instantiates a client
                    vision_client = vision.Client()


                    # Path to image file (obtain from user in the cmd prompt)
                    imgnr = input("Which image? ")
                    imgnr_s = str(imgnr)
                    inputimg = "resources/input" + imgnr_s + ".jpg"


                    # The name of the image file to annotate
                    file_name = os.path.join(
                        os.path.dirname(__file__),
                        inputimg)


                    # Loads the image into memory
                    with io.open(file_name, 'rb') as image_file:
                        content = image_file.read()
                        image = vision_client.image(
                            content=content)


                    # Performs label detection on the image file
                    # Puts all the labels in a list
                    labels = image.detect_labels()


                    # Initialises variables involved in categorising the trash
                    recycle_l = ["unsorted"]
                    recycle = "unsorted"
                    category = 0
                    tried = False
                    containsFood = False


                    # Prints all the labels and their probability
                    # Creates a new list with all the categories it is able to sort
                    print("\nLabels:")
                    for label in labels:
                        print(("%s %c %s") % (label.description, ':', label.score))
                        if "food" in label.description:
                            recycle_l.append("dangerous")
                            containsFood = True
                        elif "coffee" in label.description:
                            recycle_l.append("dangerous")
                            containsFood = True


                        elif "paper" in label.description:
```

```
                recycle_l.append("paper")

            elif "plastic" in label.description:
                recycle_l.append("plastic")

            elif "can" in label.description:
                recycle_l.append("can")



            elif "bottle" in label.description:
                recycle_l.append("plastic")
            elif (("white" in label.description) and (label.score > 0.9)):
                recycle_l.append("paper")


        # Check if it tried categorising the item at all
        try:
            recycle = recycle_l[1]
            tried = True
        except:
            recycle = "unsorted"
            tried = False


        # Code tried categorising item
        # Find the most frequently occuring category
        if tried == True:
            recycle_l.remove("unsorted")
            recycle = mode(recycle_l)


        print("\nDetected possible categories:")
        for stuff in recycle_l:
            print(stuff)


        if recycle == "paper":
            category = 1
        elif recycle == "plastic":
            category = 2
        elif recycle == "can":
            category = 3


        # Check if it contains food
        if containsFood == True:
            recycle = "unsorted"
            category = 0


        print("\nIt belongs in", recycle)


    # Programmer can import sortbotv03
    # This code checks if sortbotv03 is being imported
    # Or run in the cmd line
    if __name__ == '__main__':
        sortbot()
```

Appendix 4 – Authentication of Google Cloud Vision API

To be able to use Vision API, proper authentication and set-up has to be done during the first deployment of the programme on the specific machine (Google Could, 2014). There are two ways of authentication that SortBot can use:

a. Testing the code locally (e.g. as the developer of SortBot)

During the development stage of the programme, authentication is done by running the following code in the Google Cloud SDK: `gcloud auth application-default login.` The developer will be directed to a Google log in page. A Google account with proper Vision API permissions is required.

b. Deploying the code elsewhere (e.g. as the client of SortBot)

The machine needs a Google Developers Service Account key file, which can be created from the API console on the Google Cloud Platform. This key file needs to be stored securely as it is the only copy of this key. The file can be used by running the following code (in this instance, the file is stored in the same directory as Python, else add the path to the file):
GOOGLE_APPLICATION_CREDENTIALS='./servicekeyfilename.json'

For the API to run, the machine is required to install the Google Cloud Vision module, which is done after it has been authenticated properly.

Limitations of Google Cloud Vision API (Google Cloud, 2014)

The maximum image file the API is able to receive is 4MB, which is not a problem to SortBot. The experiment was run with a Samsung Galaxy S6 phone camera, and the resulting images are all around 500KB in size. This image size has been proven sufficient in successful labeling by the API, so it is also unnecessary to implement an expensive camera in SortBot.

Additionally, there is a limit to the number of images the API can receive per second - limiting the rate of sorting the items. The Vision API limit is 8 images per second for a given project, which will not be a big hindrance to the deployment of SortBot given that there already is a physical limit in its physical sorting of items using the servo motors.

Appendix 5 – An Example Usage of Image Recognition Software

Image taken using a phone camera:

Figure: Input image, with file name input2.jpq

```
C:\Users\Sze Tyng\Documents\GitHub\python-docs-samples\sortbot>python sortbotv04.py
Which image? 2

Labels:
drink : 0.9160799384117126
coca cola : 0.8253660202026367
soft drink : 0.7828996181488037
alcoholic beverage : 0.7587748765945435
carbonated soft drinks : 0.7331763505935669
cola : 0.6243657469749451
distilled beverage : 0.6011449098587036
bottle : 0.5899755358695984
liqueur : 0.558078408241272

Detected possible categories:
plastic

It belongs in plastic
```

Figure: Output Terminal