

**EE475 Project Final Report**

**Using AI and Robotics To Entertain Cats**

Student: **Jack Storrie**

Supervisor: **Fredrik Nordvall Forsberg**

# Declaration

“I hereby declare that this work has not been submitted for any other degree/course at this University or any other institution and that, except where reference is made to the work of other authors, the material presented is original and entirely the result of my own work at the University of Strathclyde under the supervision of Dr Fredrik Nordvall Forsberg.”

# Abstract

This project aimed to create an intelligent robot that provides entertainment for cats, built using principles of robotics and AI. The product itself takes the form of a robotic toy that uses image recognition to make smart movements and evade the target (the cat) and stay within its boundary limits (avoiding the walls of the room, chairs and other objects on the floor that may obstruct movement). The product aims to be cheap, but effective with good functionality and a durable structure.

The product will is built around a Raspberry Pi Zero computer, which will be used to allow the AI functionality to be coded using Python. It will also contain motors used to run a set of wheels for the movement of the mouse. Other low-cost components such as the basic electrical components and either a small camera or ultrasonic sensor for object recognition will be used and packaged in a shell that represents the mouse as compactly as possible. By far the most important part of this project is to have as much robotic and AI functionality fully working as possible, but aesthetic look of the product must also be considered as well. A better looking product will be more appealing to the cat and will increase the amount of engagement that the cat has with the product.

Contents

[Declaration 1](#_Toc66890405)

[Abstract 1](#_Toc66890406)

[1. Introduction 3](#_Toc66890407)

[2. Project Goals 4](#_Toc66890408)

[3. Design Stage 5](#_Toc66890409)

[3.1. Hardware Design & Component Selection 6](#_Toc66890410)

[3.2. Software Considerations 9](#_Toc66890411)

[3.3. Pseudocode Design 10](#_Toc66890412)

[3.4 Physical Design 12](#_Toc66890413)

[4. Implementation Stage 13](#_Toc66890414)

[4.1 Initial Setup 13](#_Toc66890415)

[4.2 Individual Component Testing 13](#_Toc66890416)

[4.1.1 Testing the Camera 13](#_Toc66890417)

[4.1.2 Testing the Motors 15](#_Toc66890418)

[References 18](#_Toc66890419)

# Introduction

This report will aim to give the reader a thorough understanding of this project, as well as the thought processes behind decisions made in the project, how the project progressed from beginning to end and key milestones within this timeline, as well as the technical background behind the components and techniques used in the project. An overview of the original project goals, and a review on to what degree these were achieved, will also be given.

# Project Goals

The aims of this project are split up into 3 distinct categories. Major goals, which are those that are essential for the completion of a working project, minor goals, which are not essential for an operational project but highly desirable for the completion of a high quality one, and a stretch goal that would take the project into more advanced technical areas.

The first major goal is to have the robot performing basic movement on its own across surfaces in all possible directions. Another is to implement basic AI object recognition. In its most basic form this would allow the robot to make informed movements as not to hit any obstacles, but it may not display very intelligent behaviour in terms of interacting with the cat or its general environment. The final major goal is to house all components as compactly as possible within the robot. The hardware needs protection and the robot itself has to be a reasonable size and have some sort of package or shell holding it together in order for it to be durable and structurally sound.

The first minor goal is to implement more advanced object recognition techniques to allow the robot to adapt to its environment. We want the robot to be able to make high quality informed decisions on its next movements by recognising particular objects in its vision. We also want the robot to be able to logically determine what it should do slightly ahead of time, and carry this out. The other minor goal is to create an aesthetically pleasing robot to the cat. We want the highest chance possible for the cat to be engaged with the robot, and completing this goal will ensure that happens.

The stretch goal of the project is to research and incorporate AI machine learning techniques into the robot so that it can display more intelligent behaviour by learning patterns of movement from cats each time it interacts with them. Completing this goal would allow the robot to have a database of knowledge, and use that in combination with what it detects in its current environment in order to make high quality decisions.

An evaluation of how effective these project goals were, and to what extent that the work carried out achieved them, will be included later in this report.

# Design Stage

The ‘design stage’ for this project involved considering what hardware, and software components should be used, as well as how the robot should be structured, in order to provide the best response to the project goals.

The first thing done in the consideration of the design, was to briefly research what other toys are already out there on the market for entertaining cats. This was done to give an idea of any patterns in the design characteristics of toys for cats which might be able to be applied to this project. Looking at available products, there a number of ‘catch the mouse’ style toys available on the market, such as the example shown in Figure 1, available from Australian retailer Mega Pet Warehouse[1]. This has a small model ‘mouse’ that can move around in a circular radius, moving under areas covered by plastic, and out again for the cat to catch it. This provided inspiration to make the design of this project a free-moving version of one of these catch the mouse style toys.



Figure 1: Catch the mouse style toy currently available on the market

## 3.1. Hardware Design & Component Selection

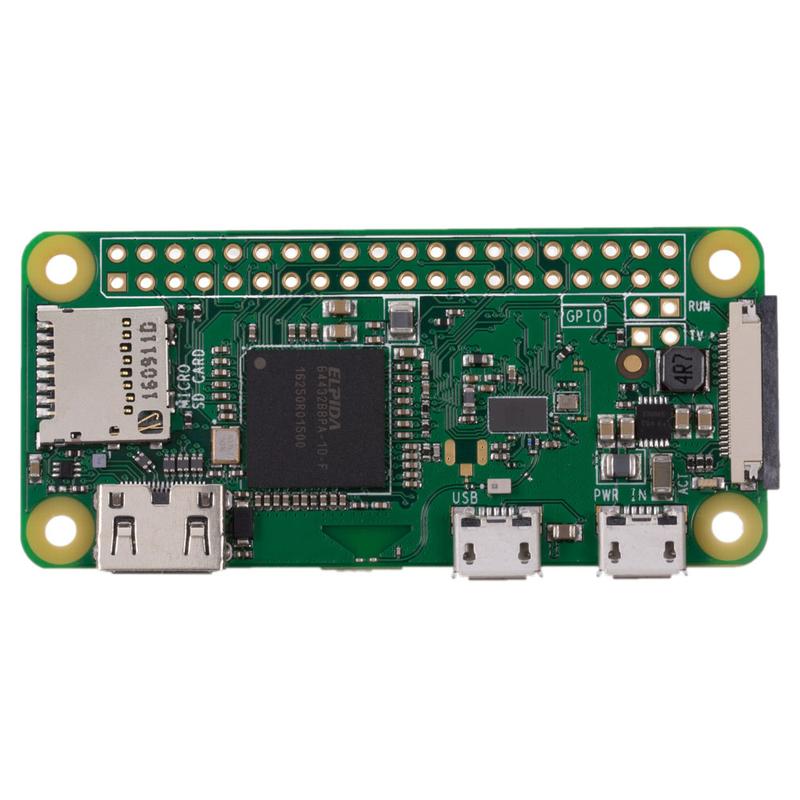
The next thing to consider was what components would be needed for the project. The main component in the project is the Raspberry Pi. In particular, the Raspberry Pi Zero W model (shown in Figure 2) was used, and this was provided to me by my supervisor. With this being provided from the start, the rest of the components would connect and build on top of this.

Figure : Raspberry Pi Zero W

In order for the robot to be able to perform image recognition, it needs a device to be able to provide visual input to it. The PiCamera, shown in Figure 3, is an attachment for Raspberry Pi that plugs directly into it via ribbon cable, with an adapter made specifically for Zero Series Raspberry Pis,, as the size of connector is slightly different. This is capable of capturing still image as well as video streams, and it is the device that allows the robot to perform object detection and image recognition.

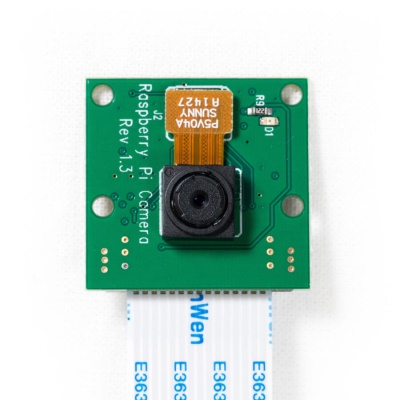


Figure : PiCamera attachment

Next, the robot needs components that will allow it to move. Intuitively, this is done through 2 motor-and-wheel sets. However, these alone wired to the Raspberry Pi don’t allow for the highest degree of control. Adding an H298 H-Bridge Motor Controller provides this extra control by allowing both motors to be controlled separately. The motors are wired to the H-Bridge, then this is wired to the GPIO pins on the Raspberry PI, allowing both motors to be commanded a single GPIO pin, and have different signals sent to them if need be. A diagram of where 2 motors connect to an H-Bridge is shown in Figure 4. A ball castor was also sourced, this is a cased metal ball that is designed to be placed between the 2 wheels of the robot. The reason for this, is to aid with friction, and easier movement across surfaces. As the robot is likely to be used on a surface such as carpet, which could potentially cause a lot of friction to the robot wheels’ rubber tires, the ball castor is a good addition to the project.

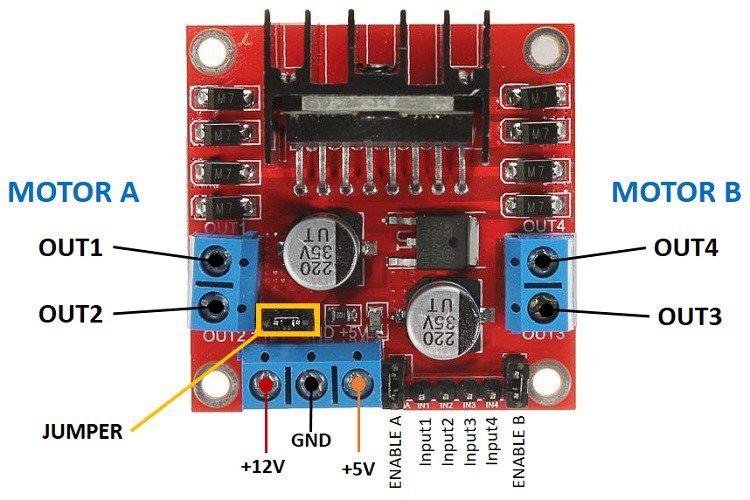


Figure : Connections on an H-Bridge Motor Controller

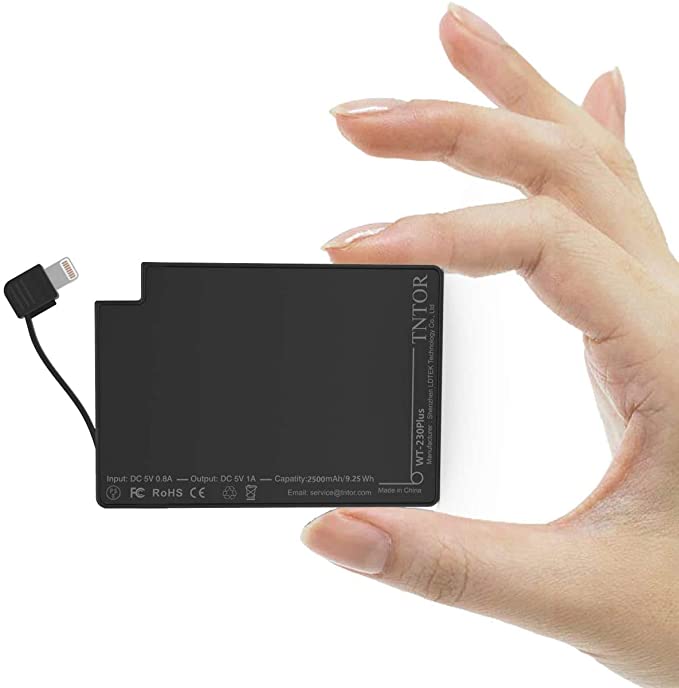
Since this is to be a free-standing project that can move by itself, the Raspberry Pi cannot be attached to a computer. It must therefore have some sort of external power source that is compact and portable enough to go along with the robot. For this purpose, a TNTOR WT-230Plus power pack (shown in Figure 5) is being used. This simply plugs into the PWR IN input of the Raspberry Pi, and provides its power while the robot is running.

Figure : TNTOR WT-230Plus Power Pack

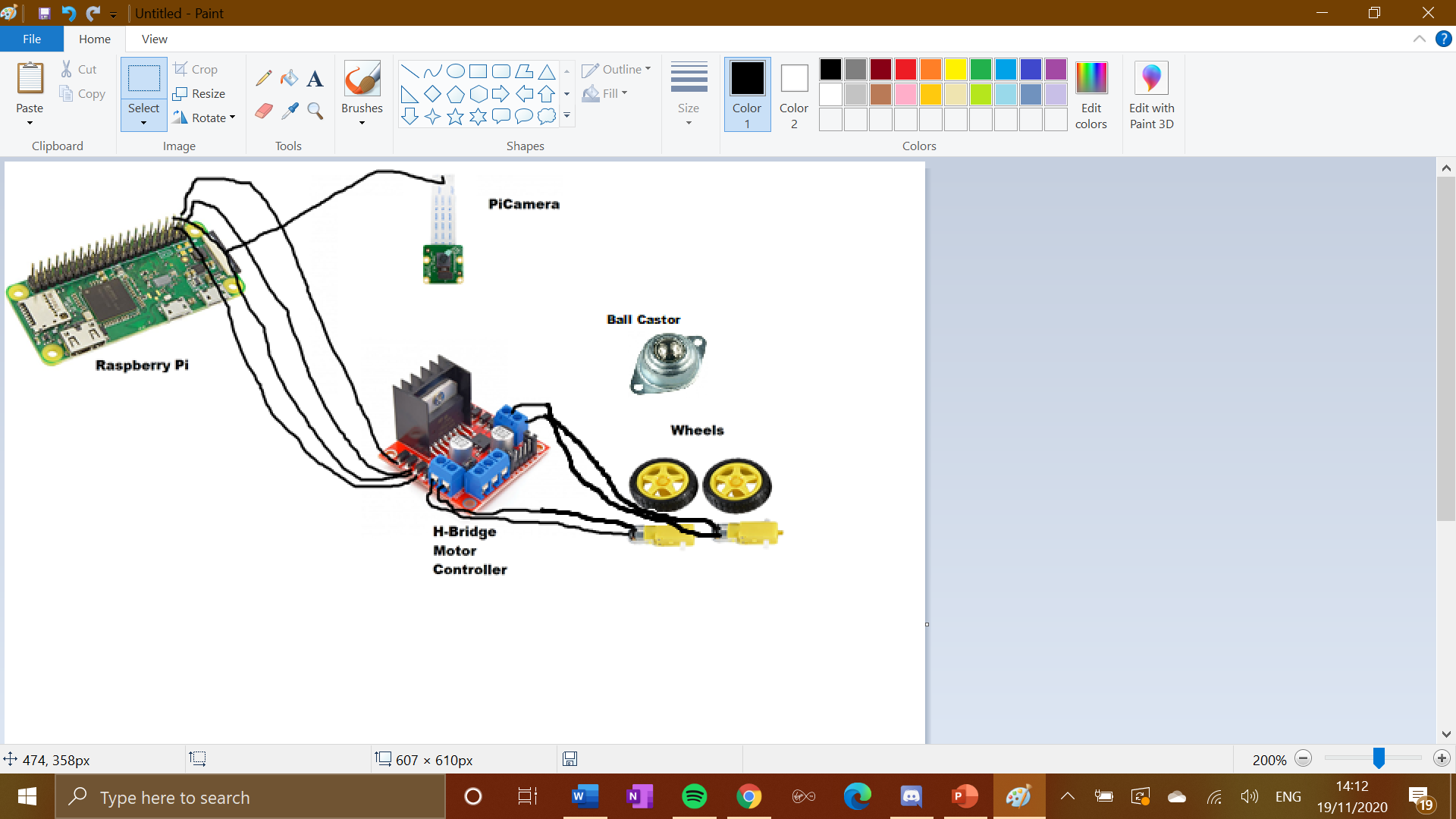
These were all the components that I chose for the original design of the robot’s hardware. A diagram I produced of how these components would be connected together roughly is shown in Figure 6. This highlights how the flow of code instructions goes from the Raspberry Pi’s GPIO pins, directly to the Pi Camera, and also through the H-Bridge which is then wired to the motors, which subsequently turn the wheels.

Figure : Original hardware design diagram

I later reconsidered the hardware design to include an ultrasonic range sensor. This is something I had originally considered but not put into the design, However, with the camera only seeing in one direction, the robot’s ability to make intelligent decisions would be extremely limited by ‘tunnel vision’. Therefore an ultrasonic range sensor was added to the original design. It is a standalone attachment, that is wired directly to the GPIO Pins on the Raspberry Pi. This allows the robot to have some sense of depth perception, to let it know if it is approaching any obstacles in any direction, rather than just in the direction that the camera is facing.

After the ultrasonic range sensor was added to the design, I believed I had included all the components I required in order to allow me to build a robot that is capable of making intelligent decisions.

## 3.2. Software Considerations

A stipulation of the project was that the python programming language is used to code the robot’s software. Python is an ideal language for this project as it handles AI functionality very well.

The main consideration in the programming of the robot was how to turn what the robot sees through the camera into instructions on where it should move. The GpioZero Python library allows hardware to be associated in code with specific GPIO pins on the Raspberry Pi. This is utilised in this project to allow signals to be sent to the wheels through the motors via the H-Bridge. A PiCamera Python library also exists, which helps define the camera’s functionality and allows it to be handled directly within the robot’s Python code.

When researching object recognition techniques, I discovered an article covering an OpenCV deep learning object recognition library for Python[2]. This has many pre-defined objects in its database that it can instantly classify given a stream of visual input. Some of these are particularly helpful in this application, including “cat” which will be the most essential. Other objects in the OpenCV database that are helpful include “chair”, “table”, “door”. What makes this so useful is that as well as classifying objects that it recognises, it gives them a “box” which is a square area roughly the size of the object with a small margin of error. This allows our robot to then move towards avoid a certain object by tracking its box. The way in which this library classifies an object is by comparing any objects it detects in the frame, against its own object database. Any matches are given a degree of confidence, and by employing minimum threshold on the degree of confidence of a detection, we can decide whether or not to take any action according to that object.

## 3.3. Pseudocode Design

After researching how the OpenCV database could be applied to this project, I decided to create a preliminary pseudocode software design to outline the operations and possible functions I believed would be necessary within the program. I decided that visual input should be taken by the PiCamera using a video stream loop, similar to that of CCTV. Taking still images at regular intervals was another possible approach, but it would be difficult to take the image ,analyse it, and choose an action before the state of the robot’s field of view had significantly changed. Using video allows for decisions to be made in real time. Using a CCTV-style video loop instead of constantly recording saves storage space, and allows the program to reset its state at regular time intervals (e.g. the start of each video loop). I decided that once an object is recognised, it should be stored in a temporary database for the remainder of this video stream loop. Once the object detection algorithm has run its course, the objects detected in frame and their location is then passed to the movement algorithm, which will use this information to determine what the most logical next movement step for the robot should be, and sends appropriate signals to the motors for this.

The pseudocode design is shown below:

define camera and motors as variables

main\_function():

camera\_setup()

while(running):

camera\_processing(video\_stream)

movement\_algrorithm(detections)

return

camera\_setup():

activate\_camera

set\_up\_video\_stream(definelengthofloop)

start\_video\_stream

return video\_stream

camera\_processing(video\_stream):

define variable for objects detected called detections

define variable for locations of detected objects called coordinates

scan frame for objects

add objects detected to detections

find location of detections in frame

add locations to coordinates

return detections, coordinates

movement\_algorithm(detections):

scan through object locations

decide on next direction signal to motors

send directional signal to motors

return

## 3.4 Physical Design

When considering how the components should be physically brought together, the main factors to think about are the size required to fit all components in, durability and workability of the material. Since I had no access to 3D printers or anything of the sort, a DIY solution that could be built in the home for the shell of the robot had to be considered. I found a pyramidal cardboard box in the house, with flaps at the top to open. This box is made out of thick cardboard that I confirmed through testing does not bend or tear under stress or pressure. The box was of sufficient size to hold all components too. Two holes could be drilled in the sides of the box in order for the wheels to stick out. Here I will have to make sure there isn’t any friction in between the side of the box and the motor-wheel attachment, which could potentially limit movement of the wheel. Another hole should be drilled into the front of the box to allow the camera to stick out. As this is connected to the Raspberry Pi via ribbon cable, this cable could be secured to the inside of the box so that it stays in place, with the camera lens itself poking out of the hole. An additional two holes should be drilled in the front of the box, in order to allow the two echoers of the ultrasonic sensor to be on the outside, and able to send out their signals to detect objects. The ball castor will be secured to the bottom of the box on the middle of the front side, and the wheels placed towards the back, this allows the whole box to lean forwards onto the ball castor, allowing free movement. The flaps will not be fully secured down, as access to the battery pack will be required.

# Implementation Stage

## 4.1 Initial Setup

The first thing that had to be carried out after the arrival of the project components is configuration of the Raspberry Pi computer. I had some issues setting up the Raspberry Pi with my laptop. Admittedly, I did expect it to be a bit more “plug and play” than it actually was. I expected the initial setup to be quick with no issues, but it actually turned out to be quite time-consuming with several issues coming up during the process. I found several potential solutions but there turned out to be reasons why I they would not apply to me. Using an ethernet cable wasn’t a viable solution as my laptop does not have a port for ethernet connections just as most modern laptops don’t. Connecting by Wi-Fi to the Raspberry Pi turned out to be an issue since I lived in shared student accommodation and don’t have access to the physical router. I then went through the process of connecting the Raspberry Pi to my laptop using USB in order to create a physical connection and configure the Pi that way. Everything was done correctly in accordance with several online tutorials, but the Raspberry Pi was not being picked up by the device manager on Windows. I changed to using a different USB cable, and this solved the issue, and I was able to SSH in to the Raspberry Pi using PuTTy. I then configured the Raspberry Pi, ensuring all the latest drivers were installed for all the functions I will need to use, including installing the Python manager and upgrading it from Python 2.7 to Python 3.7. Python packages that needed to be installed include the previously mentioned “picamera” and “gpiozero” as well as others that are used by OpenCV such as “imutils”. However, the picamera module for python was refusing to install for Raspberry Pi. A suggested solution for this was to add Google’s DNS server to the /etc/resolv.conf file, as the issued from installations usually come from failed connections to the Raspbian archives where the modules are stored (Raspbian being the Raspberry Pi exclusive operating system that has to be installed onto the Pi on initial start-up). However, when I tried to do this, it said the /etc/resolv.conf file was a “read-only file system”. When I tried to delete the file in order to remake it, it prevented me from doing this for the same reason. After researching this on online Raspberry Pi forums, it indicated an issue with the SD card partitioning. I took the SD card out of the Raspberry Pi and put it back into my laptop, and it recognised it as an unformatted SD card, despite the fact that it had previously been formatted to hold the Raspbian OS. My plan was to reflash the Raspbian OS onto the SD card (which had been previously done before starting the Pi using BalenaEtcher) and install everything back onto the Pi again through SSH. However Windows was not able to complete the formatting of the SD card. I found a solution of doing this manually through command prompt, but this gave me the error “Invalid media or Track 0 bad - disk unusable. Format failed.”. I had to source a new SD card in order to make progress with the project. There was some data loss but it was fairly minimal. This was mostly mitigated by regular uploads to the Git repository I had for the project. After a new SD Card was sourced, before work could resume on the project there is there was the quite lengthy process of reconfiguring the Pi to the state it was in before with the other SD Card. This time I set up a GUI desktop on my laptop for when I am working on the Pi by USB connection. This is done by remotely accessing the Raspberry Pi desktop through VNC Viewer. To do this, the VNC connection had to be opened on the Pi. I then installed Thonny onto the Raspberry Pi. This is a Python IDE for the Raspberry Pi desktop. This allowed me to work on the robot’s software directly within the Raspberry Pi desktop itself. The files could then be transferred through FileZilla between my laptop and the SD Card on the Raspberry Pi, in order to ensure that there was always a backup of all files that were being worked on at the current moment.

I then set up the Raspberry Pi desktop to boot to desktop when I connect through SSH automatically.

## 4.2 Individual Component Testing

The next stage was then writing and performing tests for the major peripherals. Testing out the individual components to make sure they can perform at full functionality is important, as it rules out a large number of potential issues when it comes to debugging at a later stage, and when all components are connected together.

### 4.1.1 Testing the Camera

For the camera, 3 basic test functions were written to ensure full operation. The first was to capture a still image, outputting an image file names ‘foo.jpg’. The code for this test can be seen in Figure 7. I then ran the test and the captured image can be seen in Figure 8. The camera was facing upwards at the time so it only captures my wardrobe and ceiling, as well as the smaller object of the smoke alarm. However it still shows that the camera captures image with decent enough quality to make out the objects that are in its view.

Graphical user interface, text, application, email

Description automatically generated

Figure : Code for testing capturing a still image

A dark room with a window

Description automatically generated with low confidence

Figure : Image captured during test

Graphical user interface, text, application, email

Description automatically generatedThe next test was to capture a short video. The picamera captures video in the .h264 file format. The code for this test can be seen in Figure 9.

Figure : Fixed-length video capture test

The final test of the camera, and the most crucial one for this project, was to test the camera’s ability to capture circular stream video. The code for this test can be seen in Figure 10. What happens here is a ring-buffer is used to restart the stream and overwrite the previous loop. The last n seconds of captured stream video is held on the Raspberry Pi’s memory, with n being determined by the size of ring buffer Graphical user interface, text, application, email

Description automatically generatedimposed on the stream as well as the bitrate of the video.

Figure : Stream test code

The most attention was paid to the looping stream video as this is the method that was determined to be best for the project to aid the movement of the robot during the design stage.

### 4.1.2 Testing the Motors

When the motors and wheels were set up to be tested, the motors were not receiving enough power to move the robot. This meant that the design of the robot had to be reconsidered to incorporate more power. To solve this issue, AA batteries, as well as a 4xAA battery holder with wire connections to supply extra power to the motors to allow them to run was added to the project. The implications on the design as a whole are fairly minimal, it just means a simple re-wiring job, with the ground connection coming from the H-Bridge being taken out of the Raspberry Pi, and being connected to the black wire of the battery holder. The red wire of the battery holder is then connected to the +12V connection on the H-Bridge. These 4 AA batteries then provided enough power to allow the motors to drive, and in turn should allow all components in the robot to operate together at the same time, as all the power from these goes directly to the motors, instead of power first flowing through the Raspberry Pi, and other components to get there. For the motors, functions were written and carried out for varying tests of movement for the motors and wheels, namely, the motor action required for moving in a square pattern, and moving in a continuous circle. The aim was to test the ability of the motors individually before the project moves on to a more advanced stage. These two tests demonstrated that the motors and wheels could turn in all required directions. First was the square movement test, the motors were first instructed to motion forward, before stopping, then instructed to move to the right 3 times, completing the square. The code for this test can be seen in Figure 11

Graphical user interface, text, application, email

Description automatically generated

Figure : Code for square movement test

The circular movement test first starts motioning the motors forward similarly to the square movement test, but then enters a loop of steering right continuously in the ain of producing a smooth circle with the wheels. The code for this test can be seen in Figure 12.

Graphical user interface, text, application, email

Description automatically generated

Figure : Code for circular movement test

On discovering that the main components of the robot were fully operational as they should be, the next stage of the project was to start developing the software structure outlined in the pseudocode design that will contain the movement and object recognition algorithms and all other code for the operation of the robot.

## 4.3 Putting the Whole System Together

Contrary to what I had previously thought, in order to properly secure the wires to the motors, access to a soldering iron was required, therefore the thought was that it would be difficult to put together and test the full model until access to labs was allowed. It was possible to confirm the working of the motors both individually and together using the H-Bridge, just not with the full system. When everything was in place for parts to be soldered and code tested on the system as a whole, it was clear that access to the university labs was not going to be available. To combat this, I was able to borrow a soldering iron from a family member with a background in electronics to use. Before soldering, the correct safety considerations had to be taken into account. I went back to the safety videos and read the documents that could be accessed through the first year soldering lab tutorial we were given. Reviewing all of this relevant documentation and reminding myself of all safety considerations before carrying out any soldering work on the project ensured that this work would be carried out in a safe manner. After the soldered joints had set on the motors, this meant that all the wires between the motors, H-Bridge, and battery pack were fully secured in place. After this, the wires going from the H-Bridge to the Raspberry Pi, and from the ultrasonic sensor to the Raspberry Pi, still had to be secured. When the components were picked up and moved, some of the wires would sometimes shake lose and either come out of their casings, or detach from the GPIO pin or ultrasonic sensor pin completely. Obviously, this cannot happen when the robot is in operation as it would lead to it shutting down mid-use, so these wires had to be secured to prevent them from becoming loose in situations like this. In order to do this, I removed the wires, and applied glue to the black wire casings, as well as to the base of the pins on the Raspberry Pi where the wires were being connected. This was left to set overnight, and after that, all of the wires were secure in place. The components could mow all be moved around without any wires coming out of place, meaning the full system was ready to be tested together. An additional setting that needs to be defined in order to test the system as a whole, is ensuring that the program automatically boots when the Raspberry Pi is powered on. This is required as previously, tests had been run by manually starting them from the desktop on VNC Viewer. For the robot to be free-moving, the Raspberry Pi must be programmed to start the robot’s code as soon as power is provided to the Raspberry Pi. I decided that a buffer should also be added to this in order to allow time for the robot to be placed down on a surface before the program starts running. A buffer of about 5 seconds should be sufficient to allow this. The method in which I did this, after consulting some tutorials[3] on different possible ways on doing it, was to modify the rc\_local file in the Raspberry Pi’s ‘etc’ folder. This stands for “run control” script, and its role is to determine the state in which the system should be placed at startup. The rc.local script is executed after all of the normal system services have been started (including networking, if enabled) and just before the system switches to a multiuser run level (where you would traditionally get a login prompt). While most Linux distributions do not need an rc.local, it's usually the easiest way to get a program to run on boot with the Raspbian OS. To instruct the Raspberry Pi to run the Python script on startup only takes 1 line, just a python command with the location on the Pi. This can be seen below in Figure 13. The time buffer described earlier does not need to be specified here, as it can be applied using time.sleep() within the program itself.

# 

Figure : Edited /etc/rc.local file in order to auto-run robot program at startup

# 4.4 Code Implementation

When initially writing the code for the robot, I closely followed the pseudocode designed shown previously in this report as closely as possible. As with any initial design, change and adaptations to the situations you find the program in are inevitable during the development of a program, this section will describe both my initial intentions for the program, as well as any alterations to the design that has to be made afterwards.

## 4.4.1 Packages & Globals

This section is a brief summary of the Python packages that are used in the program. The first of these is argparse. This is a package with functionality to handle command line arguments at runtime. This may seem like something that isn’t needed for this project, as the robot is free-moving and does not take any input commands while it is running. However, this package is required as part of OpenCV to calculate the confidence of correct object detections. Time is a basic Python package that contains time functionality, in the case of this project, the sleep function from this package is required. CV2 is a package created by the OpenCV authors, and it contains many functions used during the scanning of visual input in order to recognise objects. Functionality required from this includes creating the bounding box around recognised objects and locating co-ordinates on the frame for this. The imutils package includes functionality that allows a video stream to be set up in the correct format so that stills form this stream can be converted to the correct format to be analysed by the neural network in OpenCV object recognition. picamera and gpiozero are Raspberry Pi specific packages which allow all other functionality in the code to be carried out in terms of a robot based around the Raspberry Pi computer. The imports as well as global variables can be seen below in Figure 14.

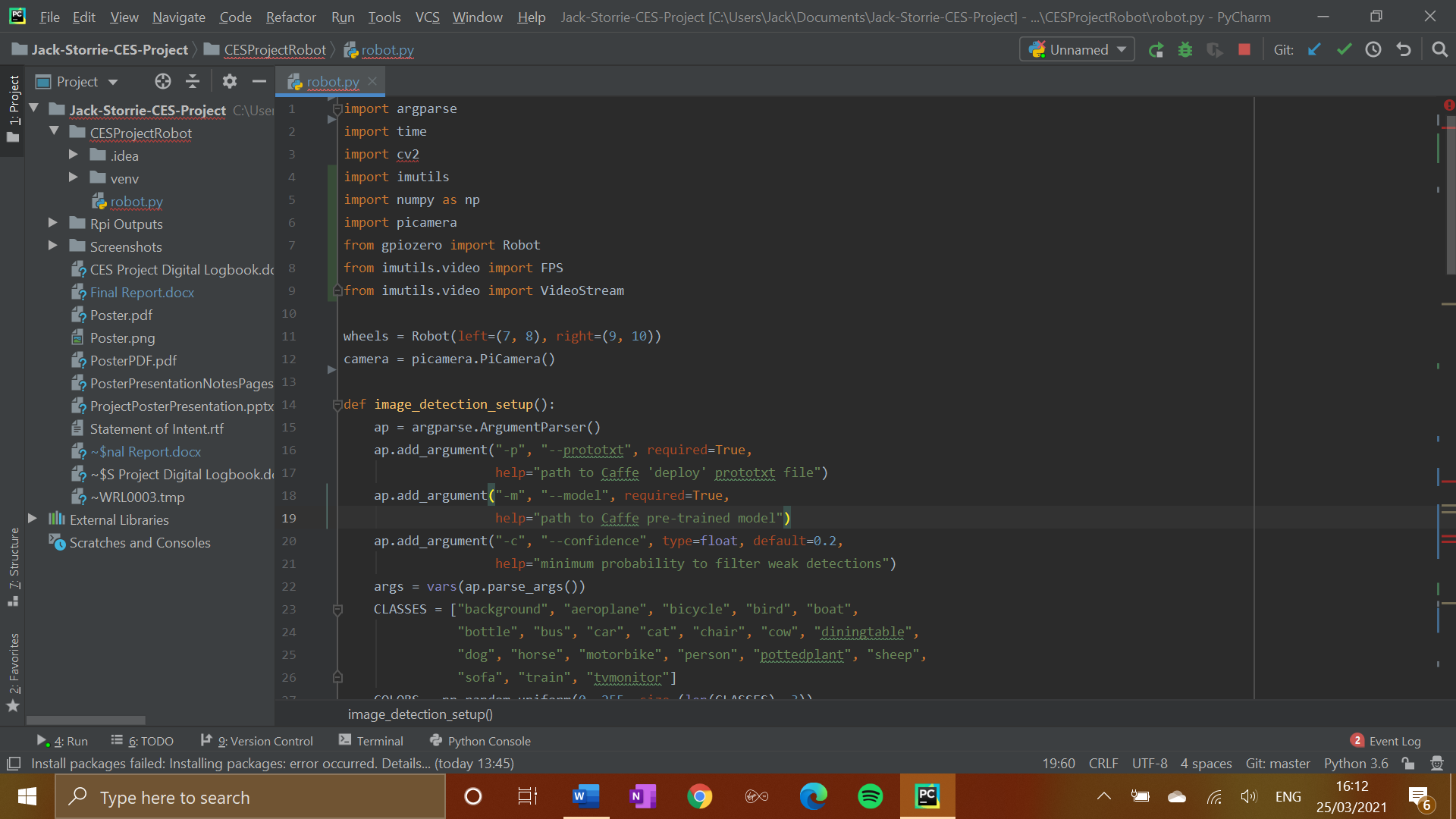


Figure : Packages imported and globals

The two global variables, ‘wheels’ and ‘camera’ are set up in this way so that the program knows that these parts of the Raspberry Pi are bring used for these roles at all times. The ‘wheels’ variable uses the gpiozero packages’ Robot function in order to permanently associate the left wheel of the robot with GPIO pins 7 and 8 on the Raspberry Pi, and the right wheel with pins 9 and 10, matching the way they are wired up physically. The ‘camera’ variable defines the PiCamera as the device capturing visual input for use throughout the program.

## 4.4.2 Main Method

The main method begins by initiating the variable ‘status’. This variable is used int the program in order to determine whether the program should continue executing (1) or stop executing (0). This variable is used within the while loop later on in the method to ensure that the cycle of object detection, decision making, and movement continues as long as the robot is switched on. In cases where the robot has to shut down, this variable will be set to 0, and the loop will terminate, ending the execution, and operation of the robot shortly after. It then sets the Boolean variable ‘stationary’ to True. This will apply every time as the motors will always be stationary upon the start-up of the robot. This is then used to inform the movement algorithm that the robot isn’t moving at this time. It then sets up the camera to record the video stream loop needed for object recognition, and runs the function that sets up the database for image recognition, among other camera settings. The tuple returned from the image detection setup algorithm is placed into a set of variables which are passed as parameters into the image scanning algorithm. The time buffer described in the last section is then implemented. This originally happened first, before anything else happened in the program, but the time taken for the steps before it is very small, and no movement of the motors is caused by anything before this line. The while loop is then entered, with the main method using the return from the scanning algorithm as a parameter ‘detections’, which is then fed into the movement algorithm. The ‘stationary’ variable is also passed into the movement algorithm as a parameter here. The main method is also responsible for ensuring the video stream is shut down, and the camera stopped before the program stops executing. This is to try and prevent any damage to the camera by a sudden unexpected loss of power when the program finishes its runtime. This main method code is seen below in Figure 14.

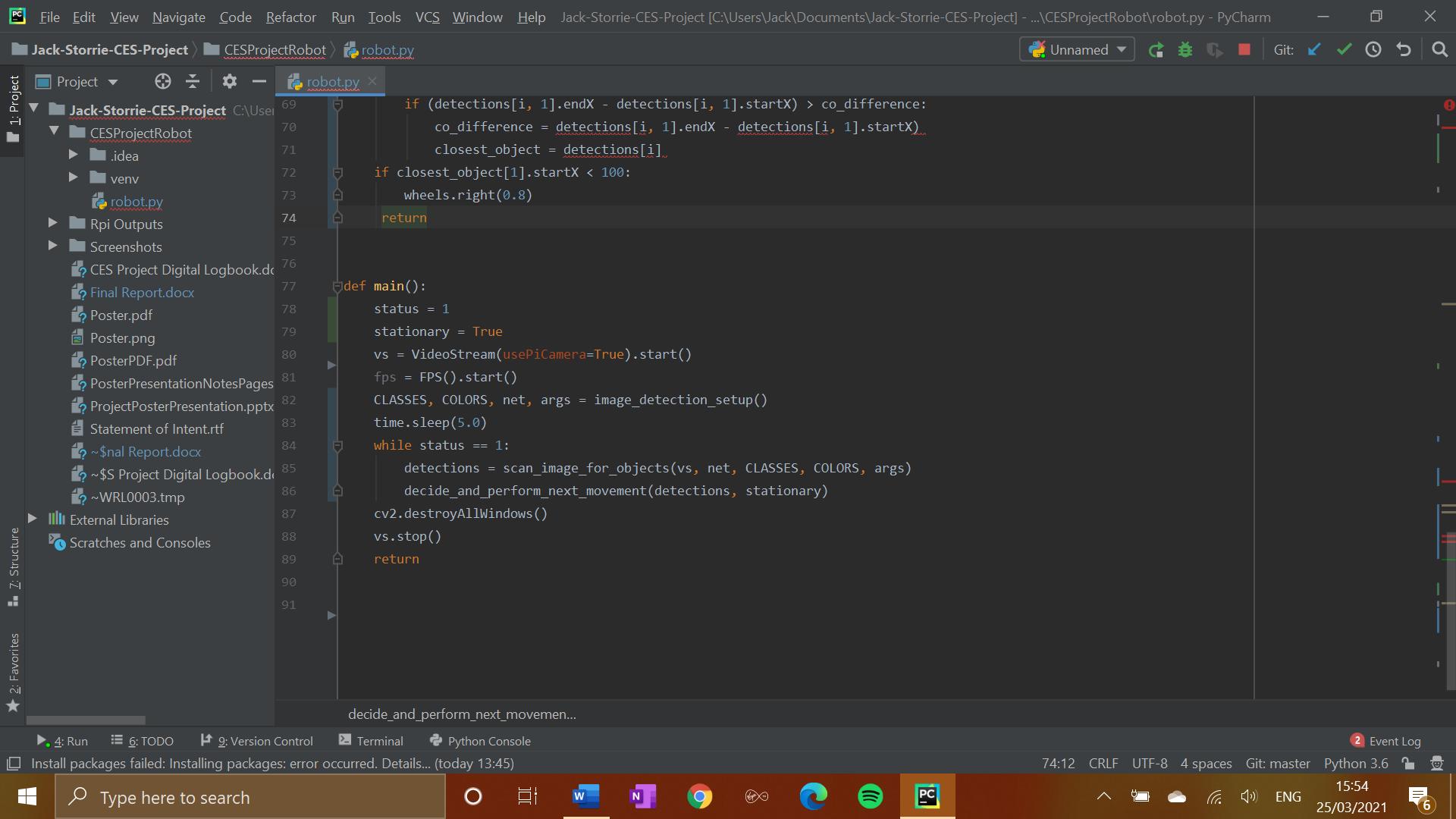


Figure : Main function of the program

## 4.4.3 Image Detection Setup

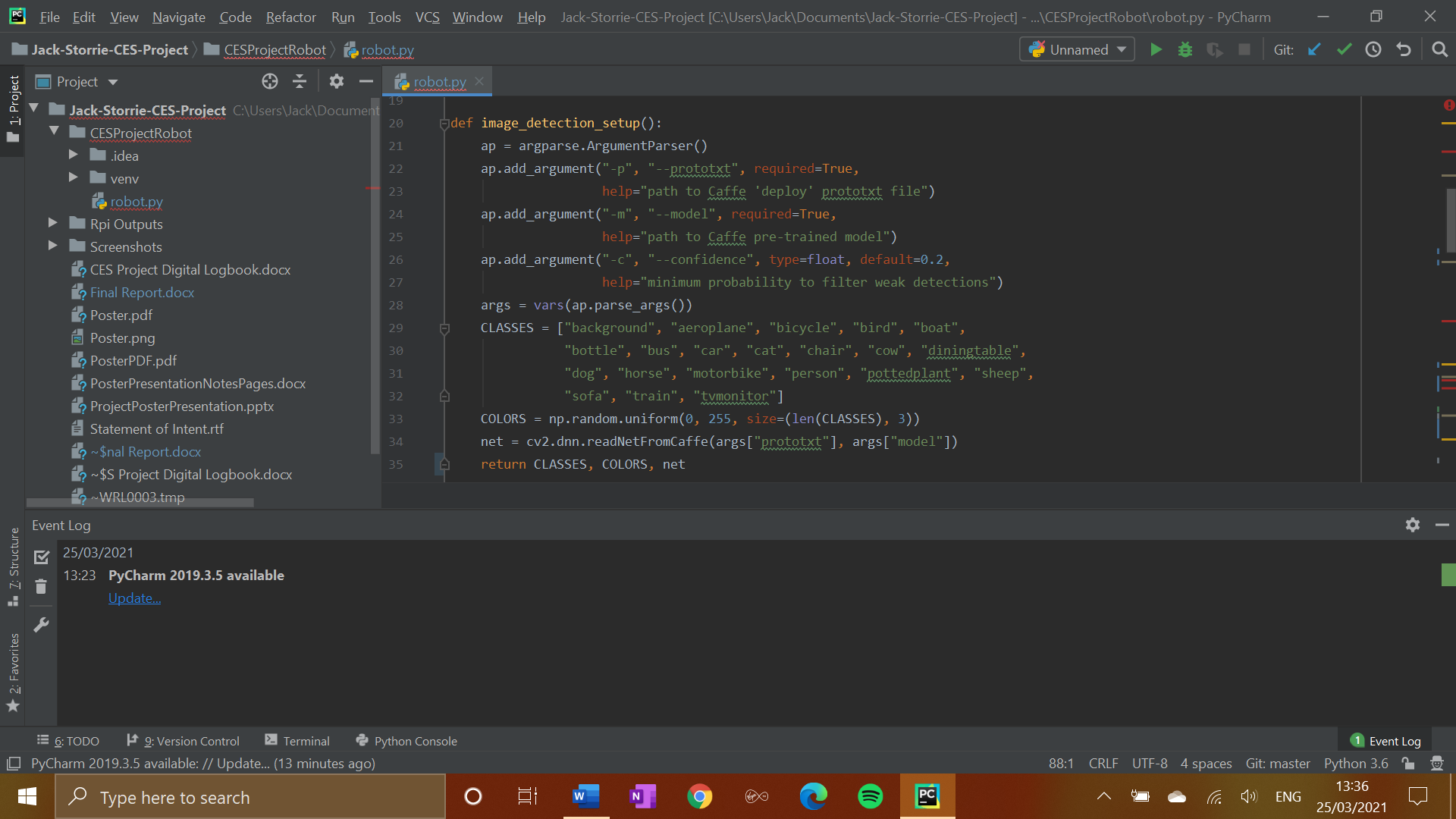
The purpose of this function is to set up the OpenCV image detection database so that it can be scanned consistently later on for matching objects in the camera’s frame. The variable ‘CLASSES’ is initialised, containing all class labels for every type of object that can be recognised by the OpenCV framework. The variable ‘COLORS’ is also initialised here, this is what is used to determine the size and location of the bounding boxes for recognised objects. The final important variable initialised here is ‘net’. By initialising this variable, the OpenCV serialized neural network model which helps to determine the identity of objects spotted by the camera is loading into the program remotely. ‘CLASSES’, ‘COLORS’ and ‘net’ are all returned by the function. to be utilised in some of the other functions in the program. The function in full is seen below in Figure 16.

Figure : Image detection setup function

A consideration that was made was to have this code within the main function itself, but in keeping with good code design principles, I gave it its own function in order to keep the main method as brief as possible, and commit to making the program as modular as possible.

## 4.4.4 Scanning Image For Objects

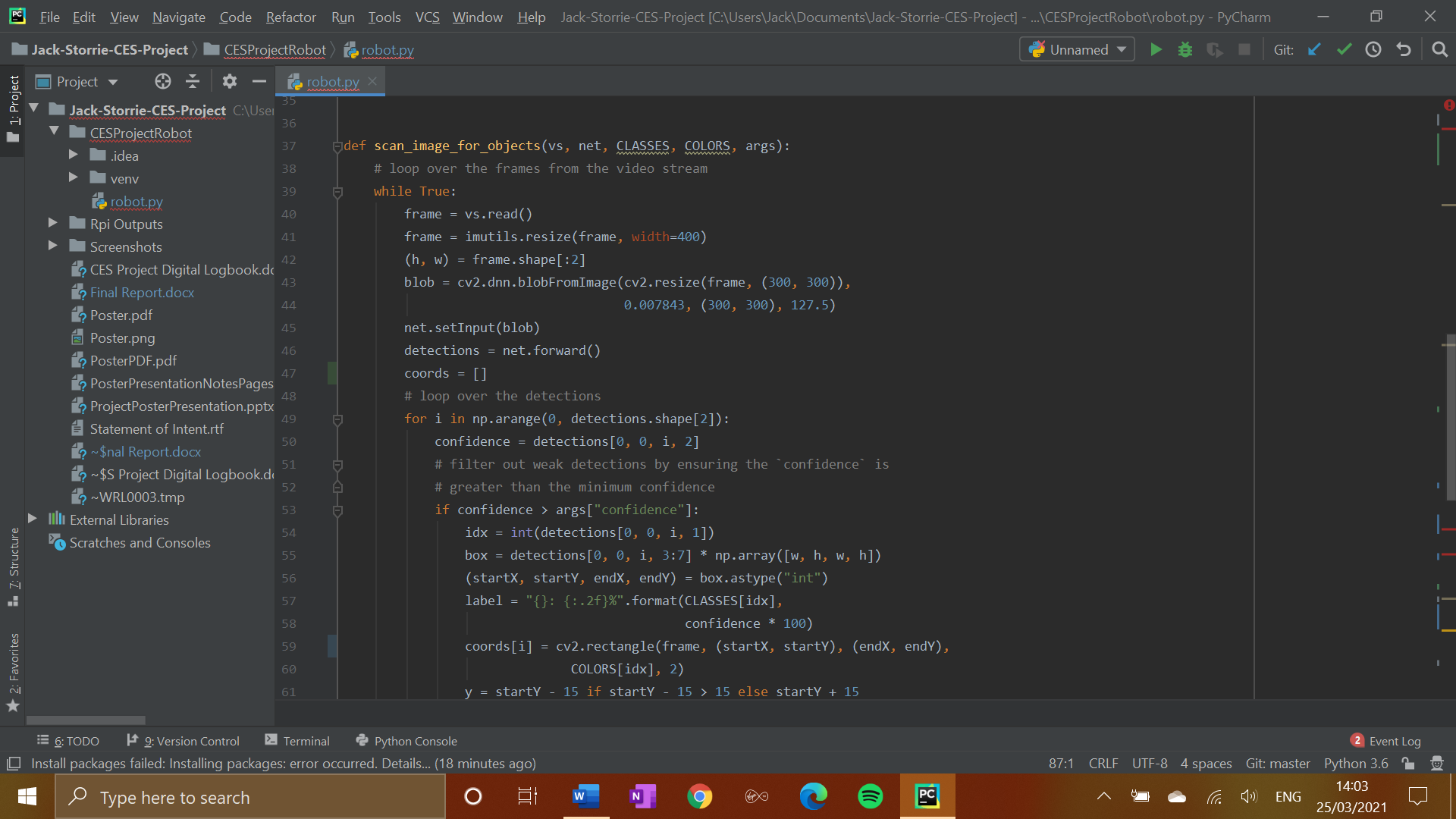
The purpose of this function is to iterate through frames of the current video stream loop, and identify any objects matching those in the OpenCV database. This method is what will identify the cat when it is visible on camera, as well as potential obstacles such as chairs and tables. The first section of the function takes the current state of the video stream and places it in a variable called ‘frame’. The frame is then resized to fit the standard size that OpenCV accepts, and this is then converted to a ‘blob’ object. A blob object is the format that is accepted by the neural network. The blob containing the video stream frame is then passed through the neural network, obtaining both the detections and bounding box predictions. A list for the current bounding box coordinates is also initialised. The first part of the function in full can be seen in Figure 17.

Figure : First part of function to scan current image for recognisable objects

After this the detections are looped over in order to identify what objects were detected in the frame, and these objects are put in a list called ‘detections’. The next section of the function again loops over the detections. For each member of the detections list, a degree of confidence in the accuracy of the identified object is calculated, in order to ensure the corresponding probability of the detection is above the predefined threshold. If it is, then the type of object detected (from ‘CLASSES’) is extracted and co-ordinates for the object’s bounding box are calculated. The co-ordinates of the bounding box for this detection are then added to the ‘coords’ list. This is a useful way to inform the robot on where to move as we have the ‘StartX’, ‘StartY,’, ‘endX’, and ‘endY’ variables that can give a pretty accurate location of where the object being detected is in reference to the direction of the robot. A list called ‘sorted\_detections’ is then initialised. The purpose of this is to put both the detections and their locations into a format that can be easily passed back to the main function. and easily parsed by the movement algorithm afterwards. To fill the ‘sorted\_detections’ list, there is a simple for loop, which takes each detection and the corresponding co-ordinates, and puts them in a 2tuple. This makes the ‘sorted\_detections’ list an array of 2tuples containing all objects currently in frame and their co-ordinates, from which a general location can be extracted and worked out later on in the scan-movement cycle of the program. The list of sorted detections is then returned by the function, to the main function to be passed on to the movement algorithm. The second part of the function to scan the image for objects can be seen in full in Figure 18.

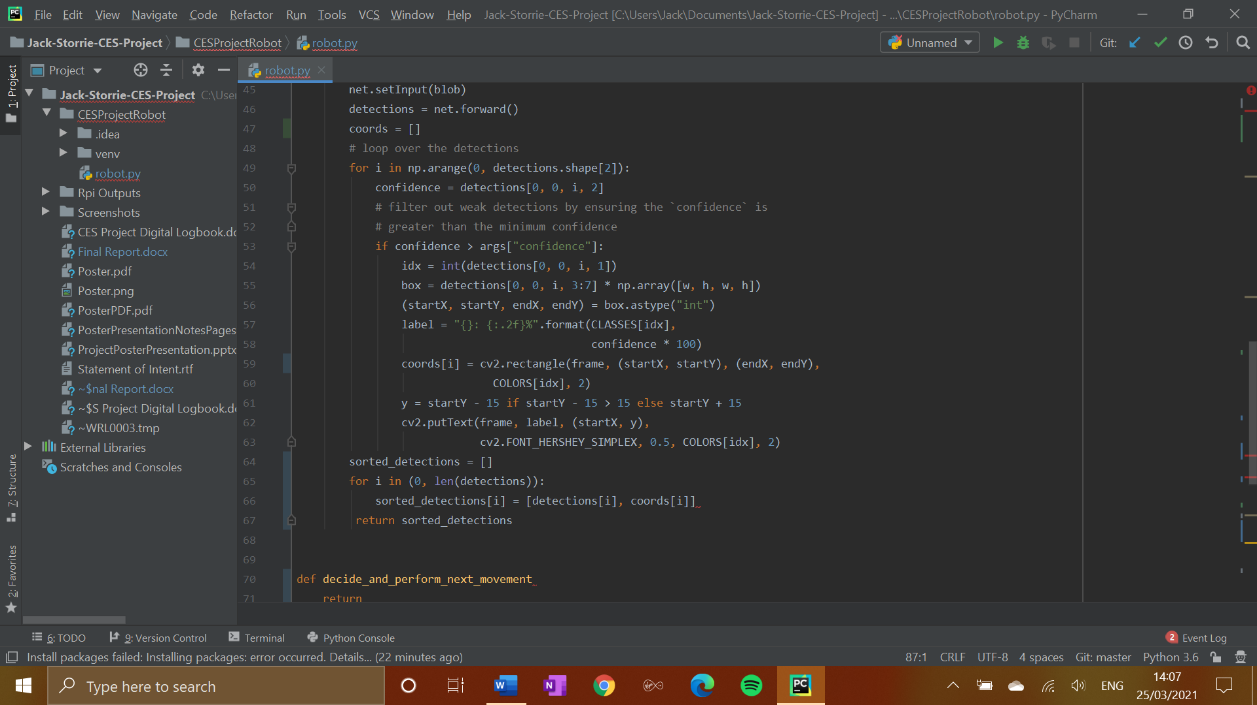


Figure : Second part of the function to scan the image for objects

## 4.4.5 Movement Algorithm

The movement algorithm is where the program takes all of the image processing done in the previous functions and puts this into action. The purpose of this function is to determine in which direction the robot should make its next movement, and then have it carry that movement out. The opening line initialises the variable ‘closest\_object’ to the first member of the detections list. This is done because the detections list is going to be put through a comparative for loop in order to determine which member of the list should be assigned to ‘closest\_object’, so it os initially set to the first member of this list as a default value. It is called closest\_object as the robot will try to inform its next movement in regards to the ‘closest’ object to it, which is classified as the largest recognisable object in frame. The variable ‘co-difference’ is also initialised before the for loop as zero, as this will also be used comparatively inside the for loop. The for loop iterates across each member of the detections list. The first line in the loop initialises the variable ‘current\_co\_difference’. This is the current object’s lower x co-ordinate subtracted from its higher x co-ordinate, and represents the relative size of the object in the current frame, which is being linked to the closeness of that object to the robot. If ‘current\_co\_difference’ is higher than the current value in ‘co\_difference’, then it becomes the new benchmark for the rest of the members of the detection list to be compared to. The current object is also placed in the ‘closest\_object’ variable, before the for loop moves on to its next iteration. By finding the largest object in frame, and therefore the object thought to be in the closest proximity to the robot, that object becomes the priority object for the rest of the algorithm to focus on, and the location of this object in the frame will help decide the next movement the robot chooses.

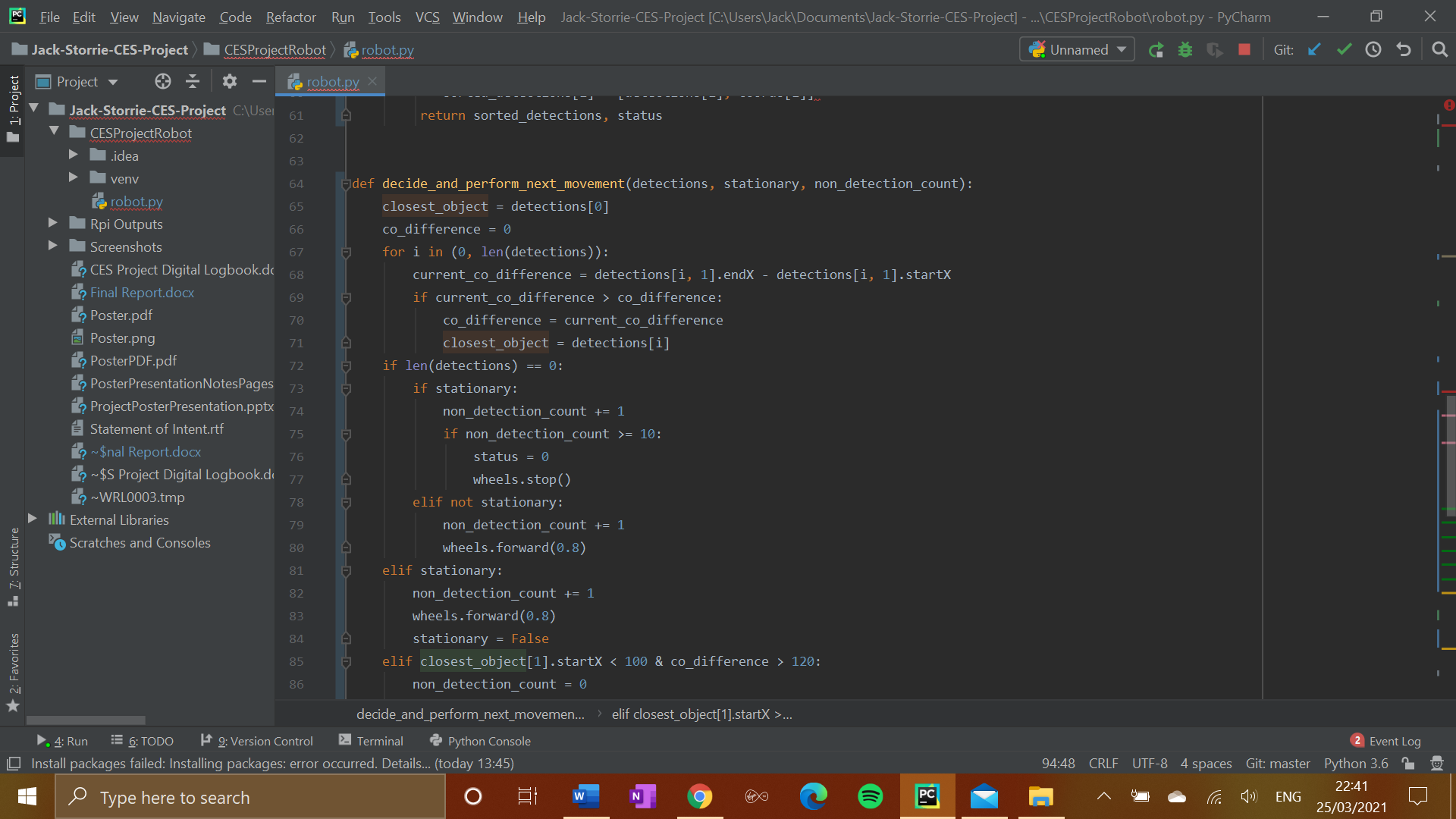


Figure : Part of movement algorithm determining the closest object

# 5. Evaluation

# References

[1] <https://shop.megapet.com.au/catch-the-mouse-motion-cat-toy-0639737307900-25941/>

[2] <https://www.pyimagesearch.com/2017/10/16/raspberry-pi-deep-learning-object-detection-with-opencv/>

[3] <https://learn.sparkfun.com/tutorials/how-to-run-a-raspberry-pi-program-on-startup/all>