



# **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 is built around a Raspberry Pi Zero W 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, a small camera and ultrasonic sensor for object recognition will be used and packaged in a shell 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 should 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
Abstract .....	1
1. Introduction .....	4
2. Project Goals .....	4
3. Design Stage .....	5
3.1. Hardware Design & Component Selection .....	6
3.2. Software Considerations .....	10
3.3. Pseudocode Design .....	11
3.4 Physical Design .....	13
4. Implementation Stage.....	14
4.1 Initial Setup .....	14
4.2 Individual Component Testing.....	16
4.1.1 Testing the Camera .....	16
4.1.2 Testing the Motors .....	18
4.3 Putting the Whole System Together .....	20
4.4 Code Implementation .....	22
4.4.1 Packages & Globals .....	23
4.4.2 Main Method.....	24
4.4.3 Image Detection Setup .....	26
4.4.4 Scanning Image For Objects .....	27

4.4.5 Movement Algorithm.....	29
5. Evaluation .....	32
5.1 Comparison With Similar Project .....	32
5.2 Testing for Evaluation .....	33
5.3 Review of Project Goals .....	35
6. Conclusion .....	36
References .....	38

## **1. 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. The report begins with an overview of the project goals, which are then reviewed again at the end with comment on how they were achieved and to what extent. The report will then take you through the design stage of the project in detail, before giving detail on the implementation of this design, including thorough analysis of the code written. There will then be an evaluation of the project as a whole, including a comparison to an academic paper with similar themes to this project.

## **2. 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. When measuring the degree of success of the project, the major goals will be the main focus of attention.

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. Ideally, the robot should be able to avoid the cat as well as obstacles in its field of view effectively. 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.

### **3. Design Stage**

The design stage of 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 could be applied to this project. When searching for available products on the internet, 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<sup>[1]</sup>. 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. The robot will act as the ‘mouse’, with the objective being that the cat should chase the robot, and the robot should attempt to avoid it for as long as possible, providing entertainment to the cat.



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

### **3.1. Hardware Design & Component Selection**

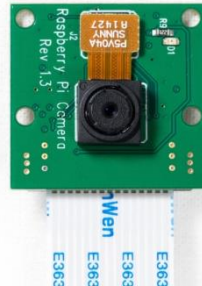
The next thing to consider was what components would be needed for the project. The most important single component in the project is the Raspberry Pi. In particular, the Raspberry Pi Zero W model (shown in Figure 2) was used. With this being provided from the start, the rest of the components would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



*Figure 2: 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 made specifically for Raspberry Pi that plugs directly into it via ribbon cable. The model of Raspberry Pi that was used in this project required an adapter cable for the picamera, as the dimensions of the input on the Pi are slightly different on Zero series models to the rest of the Raspberry Pi range. The PiCamera is capable of

capturing still image as well as video streams, and it is the device that allows the robot to scan images taken to perform object recognition.



*Figure 3: PiCamera attachment*

Next, the robot needs components that will allow it to move. The movement of this robot will be driven by 2 DC gear motors, attached to plastic wheels with rubber tires covering them. However, we desire a greater degree of control over the motor action than what is provided by wiring the motors directly on to the Raspberry Pi. Adding an H298 H-Bridge Motor Controller in between the motors and the Pi provides this extra degree of control by allowing both motors to be controlled individually. The motors are wired to the H-Bridge, then the H-Bridge is wired to the GPIO pins on the Raspberry Pi, allowing both motors to be commanded through using a set of 4 GPIO pins on the Pi, 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 component to add to the robot as it will allow for smoother, more free-flowing movement by the robot.



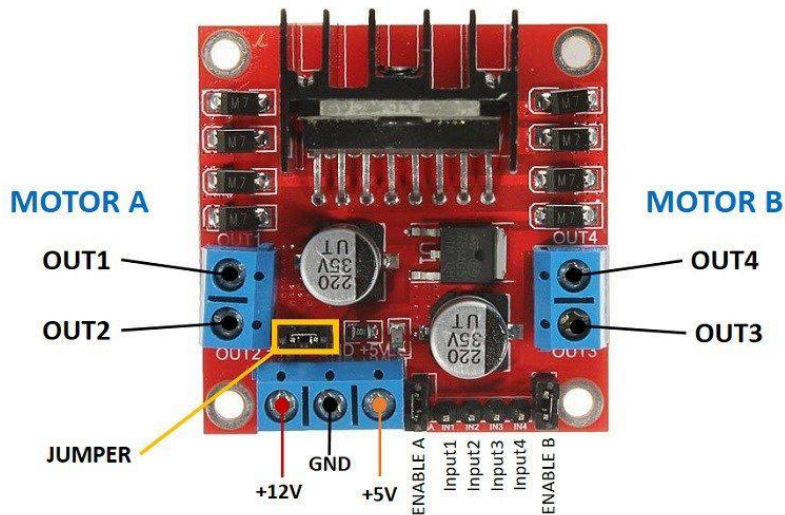


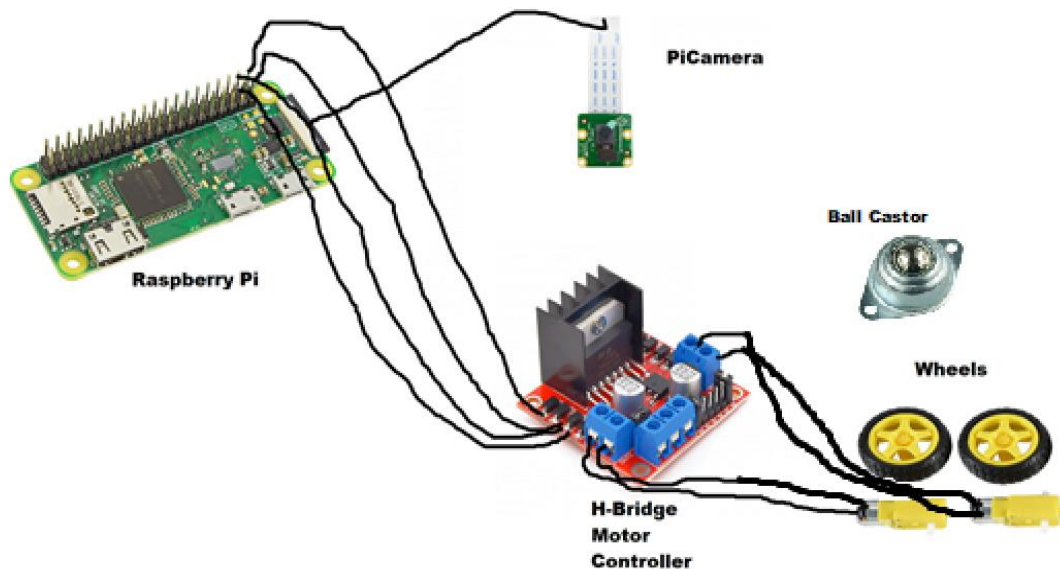
Figure 4: Connections on an H-Bridge Motor Controller

Since the robot needs to move freely on its own, the Raspberry Pi cannot be attached to a computer or other large external power source that cannot move along with it. It must therefore have some sort of power source that is compact and portable enough to go along with the robot. For this purpose, a TNTOR WT-230 Plus power pack (shown in Figure 5) is being used. This simply plugs into the PWR IN input of the Raspberry Pi, and provides the power needed directly to the Pi while the robot is in operation.



Figure 5: TNTOR WT-230 Plus Power Pack

These were all the components that were chosen for the original design of the robot's hardware. The diagram 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 6: Original hardware design diagram*

After presenting my interim poster for the project, the hardware design was revamped to include a US-015 ultrasonic range sensor. This is something that had been originally considered in the early stages of the project, but ultimately not put into the initial 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, the full hardware design was complete, containing all the components required for this project to be successful in creating a robot capable of making intelligent decisions on its own while moving through an environment.

### 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 to use for this project, due to the large number of libraries and packages that exist for it, its compatibility with both AI projects and with the Raspberry Pi, and that plenty of previous experience had been had with the language.

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. The gpiozero library has a database of many different common peripherals that are used with Raspberry Pi computers, and provides functionality to easily drive those peripherals directly from the GPIO pin rack through code. 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 technique for Python<sup>[2]</sup>. This uses a neural network called MobileNet which can be accessed using OpenCV's own 'cv2' Python package to carry out image scanning and object recognition. 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, the next part of the design stage was to create a preliminary pseudocode software design to outline the operations and possible functions that would be necessary within the program. It was decided that the visual input should be taken by the PiCamera using a video stream loop, similar to that of CCTV, since picamera is by far the most compatible peripheral to the Raspberry Pi that can capture video. 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 stills from the video stream at as quick a pace as possible. 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). 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)
```

```

        ultrasonic_scan()

        movement_algorithm(detections, distance)

    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

ultrasonic_scan():

    send out pulse

    calculate pulse time

    calculate distance to nearest object

```

```
    return distance

movement_algorithm(detections, distance):

    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 there was no access to 3D printers or any technology that would have allowed a custom body to be created for the robot, a DIY solution that could be built at home had to be considered. This came down to finding some object that could be adapted to be used for a robot body. The solution found for this was a pyramid shaped cardboard box that had previously been used to hold flowers. This box was made out of thick cardboard, confirmed through testing that it did not bend, tear or deform under a reasonable stress or pressure. The box was of sufficient size to house all of the robot's components and wires. Two holes could be drilled in the sides of the box in order for the wheels to slot out. Here it is vital to make sure there isn't too much friction in between the side of the box and the motor-wheel attachment, as this could potentially limit movement of the wheels and therefore the robot itself. Another hole should be drilled into the front of the box to allow the lens of the camera to come out and be able to see the environment and record. 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. An additional two holes should be drilled in the front of the box, in order to allow the ultrasonic sensor to be on the outside, and be able to send out its 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 of the box, which are on the top side,

will not be fully secured down, as access to the battery pack will be required. The box is a bright, pink colour which could be aesthetically pleasing to the cat. Another thing the cat may notice is that the layout of the ultrasonic sensor and camera poking out from inside the box may look like a face, giving the robot a sense of character. The ultrasonic sensor's two circular pulsars may look like eyes, and the camera lens a small mouth. This is something that doesn't have any effect on the performance of the robot, but may go towards achieving the minor project goal of having the robot look aesthetically pleasing to a cat that comes into contact with it.

## **4. 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. There were some issues setting up the Raspberry Pi with the laptop being used for the development of the project. The initial setup was expected to be very quick with no issues, but it actually turned out to be quite time-consuming with several issues coming up during the process. Using an ethernet cable wasn't a viable solution as the laptop did 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 at first since the project was being carried out entirely in shared student accommodation and there was no access to the physical router. The only method of working on the Raspberry Pi that was then viable, was connecting the Pi to the laptop via 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. A different USB cable was then tried, and this solved the issue, and I was able to SSH in to the Raspberry Pi using PuTTY. The Raspberry Pi then went through the configuration process, ensuring all the latest drivers were installed for all the functions needed for the project, 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 an attempt was made to open this file in the Raspberry Pi's nano editor, it said the `/etc/resolv.conf` file was a "read-only file system". When trying to delete the file with the intention of recreating it from scratch, this was prevented from happening for the same reason. After researching this on online Raspberry Pi forums, it indicated an issue with the SD card partitioning. The SD card was taken out of the Raspberry Pi and put back into the laptop, and it recognised it as an unformatted SD card, despite the fact that it had previously been formatted to hold the Raspbian OS. The plan from there 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. The intended solution for this was doing it manually through command prompt, but this gave me the error "Invalid media or Track 0 bad - disk unusable. Format failed.". This means the SD card was corrupt and from this point unusable, and a new one had to be obtained 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 for the project. After a new SD Card was sourced, before work could resume on the project there was the process of reconfiguring the Pi to the state it was in before with the other SD Card. After this a desktop GUI was set up in order to make working on the Pi far easier than having to do everything through command line.. 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. The Thonny program was then installed onto the Raspberry Pi. This is a Python IDE for the Raspberry Pi desktop. This allowed work to be carried out on the programming for the robot directly within the Raspberry Pi desktop itself. The files could then be transferred through FileZilla between the 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.



## 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.

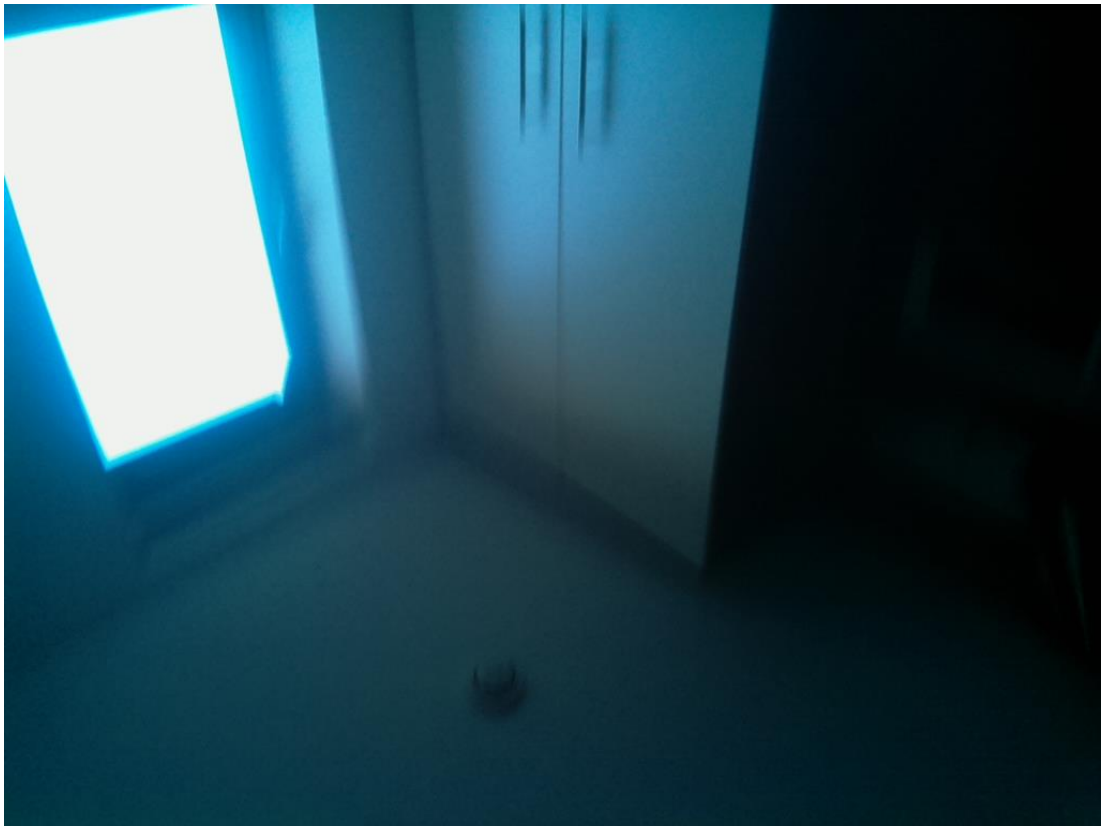
### 4.1.1 Testing the Camera

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, even if the quality of image provided isn't the highest due to the low megapixel nature of the picamera.

```
from time import sleep
from picamera import PiCamera

camera = PiCamera()
camera.resolution = (1024, 768)
camera.start_preview()
# Camera warm-up time
sleep(2)
camera.capture('foo.jpg')
```

*Figure 7: Code for testing capturing a still image*



*Figure 8: Image captured during test*

The 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.

```
py x cameratest.py x
import picamera

camera = picamera.PiCamera()
camera.resolution = (640, 480)
camera.start_recording('my_video.h264')
camera.wait_recording(10)
camera.stop_recording()
```

*Figure 9: 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 imposed on the stream as well as the bitrate of the video.

```
import io
import random
import picamera

def motion_detected():
    # Randomly return True (like a fake motion detection routine)
    return random.randint(0, 10) == 0

camera = picamera.PiCamera()
stream = picamera.PiCameraCircularIO(camera, seconds=20)
camera.start_recording(stream, format='h264')
try:
    while True:
        camera.wait_recording(1)
        if motion_detected():
            # Keep recording for 10 seconds and only then write the
            # stream to disk
            camera.wait_recording(10)
            stream.copy_to('motion.h264')
finally:
    camera.stop_recording()
```

*Figure 10: 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.

```
from gpiozero import Robot
from time import sleep

wheels = Robot(left=(7, 8), right=(9, 10))

wheels.forward(0.5)
sleep(2)
wheels.right(0.5)
sleep(6)
wheels.stop()
```

*Figure 11: 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 aim of producing a smooth circle with the wheels. The code for this test can be seen in Figure 12.

```
from gpiozero import Robot
from time import sleep

wheels = Robot(left=(7, 8), right=(9, 10))

wheels.forward(0.5)
sleep(2)
wheels.stop()
for i in range(3):
    wheels.right(0.5)
    sleep(2)
    wheels.stop()
```

*Figure 12: Code for circular movement test*

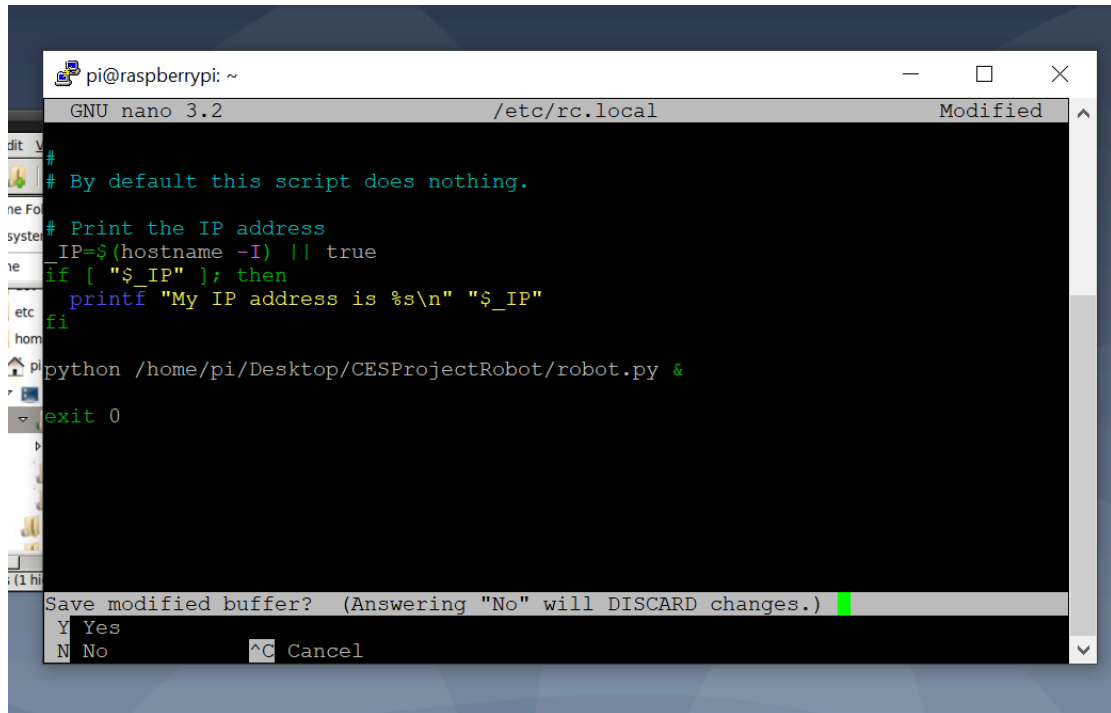
On discovering that the main components of the robot were fully operational as they should be, the next stage of the project was to start developing the software structure outlined in the pseudocode design that will contain the movement and object recognition algorithms and all other code for the operation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

### **4.3 Putting the Whole System Together**

Contrary to what was previously thought when components were purchased, 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, it became possible 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. This was done by going 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 loose 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, the wires were removed from their places, and glue applied 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 now 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. To combat this, it was 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 this was done, after consulting some tutorials<sup>[3]</sup> 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.



```
pi@raspberrypi: ~  
GNU nano 3.2 /etc/rc.local Modified  
#  
# By default this script does nothing.  
# Print the IP address  
IP=$(hostname -I) || true  
if [ "$IP" ]; then  
    printf "My IP address is %s\n" "$IP"  
fi  
python /home/pi/Desktop/CESProjectRobot/robot.py &  
exit 0  
Save modified buffer? (Answering "No" will DISCARD changes.)  
Y Yes  
N No ^C Cancel
```

Figure 13: 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, the pseudocode design shown previously in this report was followed 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. After having a separate function for the ultrasonic scan initially, the structure was changed to move the ultrasonic scan to the start of the movement algorithm instead. This reduces the number of parameters being passed about, and it also simplifies the loop within the main function, bringing it down to a simple two function cycle of image scanning then movement. The “deal with detections” phase was also removed. This turned out to tie in well with the creation of bounding boxes in the scanning function, so sorting the detections out was also done within this function. This meant that the detections list could be passed straight from the scanning function to the movement algorithm via the main function

without having to be passed as a parameter through another transitional function, which could improve the efficiency of the scanning-movement cycle.

#### **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 from 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.



```

# imports
import argparse
import time
import cv2
import imutils
import numpy as np
import picamera
import gpiozero
from imutils.video import VideoStream

# globals
wheels = gpiozero.Robot(left=(7, 8), right=(9, 10))
camera = picamera.PiCamera()
us_sensor = gpiozero.input_devices.DistanceSensor(24, 18)

```

Figure 14: Packages imported and globals

There are three global variables, ‘wheels’, ‘camera’, and ‘us\_sensor’ are set up in this way so that the program knows that these parts of the Raspberry Pi are being 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. The ‘us\_sensor’ variable uses gpiozero’s database of Raspberry Pi input devices to permanently assign the ECHO and TRIG functions of the ultrasonic sensor to GPIO pins 24 and 18 on the Raspberry Pi, also equivalent to the way in which they are wired on to the board.

#### 4.4.2 Main Method

The main method begins by initiating the variable ‘status’. This variable is used in 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 ultrasonic sensor is then set up for use, with the trigger pulse being define as an output and the echo pulse being defined as an input. 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 15.

```
def main():
    # preparing variables and performing time buffer
    status = 1
    stationary = True
    non_detection_count = 0
    vs = VideoStream(usePiCamera=True).start()
    CLASSES, COLORS, net, args = image_detection_setup()
    gpiozero.setup(us_sensor[1], gpiozero.OUT)
    gpiozero.setup(us_sensor[0], gpiozero.IN)
    time.sleep(5.0)
    # scanning-movement cycle
    while status == 1:
        detections = scan_image_for_objects(vs, net, CLASSES, COLORS, args, status)
        decide_and_perform_next_movement(detections, stationary, non_detection_count, status)
    #cleanup
    cv2.destroyAllWindows()
    vs.stop()
    return
```

Figure 15: 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. '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.

```
def image_detection_setup():
    ap = argparse.ArgumentParser()
    ap.add_argument("-p", "--prototxt", required=True,
                    help="path to Caffe 'deploy' prototxt file")
    ap.add_argument("-m", "--model", required=True,
                    help="path to Caffe pre-trained model")
    ap.add_argument("-c", "--confidence", type=float, default=0.2,
                    help="minimum probability to filter weak detections")
    args = vars(ap.parse_args())
    CLASSES = ["background", "aeroplane", "bicycle", "bird", "boat",
               "bottle", "bus", "car", "cat", "chair", "cow", "diningtable",
               "dog", "horse", "motorbike", "person", "pottedplant", "sheep",
               "sofa", "train", "tvmonitor"]
    COLORS = np.random.uniform(0, 255, size=(len(CLASSES), 3))
    net = cv2.dnn.readNetFromCaffe(args["prototxt"], args["model"])
    return CLASSES, COLORS, net
```

Figure 16: 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, this was given its own function in order to keep the main method as brief 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.

```
def scan_image_for_objects(vs, net, CLASSES, COLORS, args):  
    # loop over the frames from the video stream  
    while True:  
        frame = vs.read()  
        frame = imutils.resize(frame, width=400)  
        (h, w) = frame.shape[:2]  
        blob = cv2.dnn.blobFromImage(cv2.resize(frame, (300, 300)),  
                                     0.007843, (300, 300), 127.5)  
        net.setInput(blob)  
        detections = net.forward()  
        coords = []
```

Figure 17: 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.

```
# loop over the detections
for i in np.arange(0, detections.shape[2]):
    confidence = detections[0, 0, i, 2]
    # filter out weak detections by ensuring the `confidence` is
    # greater than the minimum confidence
    if confidence > args["confidence"]:
        idx = int(detections[0, 0, i, 1])
        box = detections[0, 0, i, 3:7] * np.array([w, h, w, h])
        (startX, startY, endX, endY) = box.astype("int")
        label = "{}: {:.2f}%".format(CLASSES[idx],
                                    confidence * 100)
        coords[i] = cv2.rectangle(frame, (startX, startY), (endX, endY),
                                COLORS[idx], 2)
        y = startY - 15 if startY - 15 > 15 else startY + 15
        cv2.putText(frame, label, (startX, y),
                    cv2.FONT_HERSHEY_SIMPLEX, 0.5, COLORS[idx], 2)
sorted_detections = []
for i in (0, len(detections)):
    sorted_detections[i] = [detections[i], coords[i]]
return sorted_detections
```

Figure 18: 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 is 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. This first part of the algorithm can be seen below in Figure 19.

```
def decide_and_perform_next_movement(detections, stationary, non_detection_count):
    closest_object = detections[0]
    co_difference = 0
    for i in (0, len(detections)):
        current_co_difference = detections[i, 1].endX - detections[i, 1].startX
        if current_co_difference > co_difference:
            co_difference = current_co_difference
            closest_object = detections[i]
```

Figure 19: Part of movement algorithm determining the closest object

The next part of the algorithm is performing a pulse with the ultrasonic sensor to gain the distance (in centimetres) of the nearest object. A 10 microsecond trigger pulse is sent out by the ultrasonic sensor. This is then followed by the echo pulse, which has its start and end times measured by the time.time function. The duration of the pulse is then calculated, and using the average speed of sound through air of 343 m/s, a simple speed, distance, time formula is carried out and rounded to give the distance to the nearest object in centimetres. This part of the algorithm can be seen below in Figure 20.

```
# performing ultrasonic sensor pulse
gpiozero.output(us_sensor[1], True)
time.sleep(0.00001)
gpiozero.output(us_sensor[1], False)
while gpiozero.input(us_sensor[0]) == 0:
    pulse_start = time.time()
while gpiozero.input(us_sensor[0]) == 1:
    pulse_end = time.time()
pulse_duration = pulse_end - pulse_start
distance = round((pulse_duration * 17150), 2)
```

Figure 20: Performing the ultrasonic sensor pulse

The algorithm then enters a large else-if construct which will go on to determine the next movement made by the robot. It starts off by searching for a couple of special cases. Firstly, if the list of detections is empty (i.e. no recognisable objects are currently in view) then 1 will be added to the 'non\_detection\_count' variable. This

variable is designed to shut down the robot if it is stationary or inactive for a long period of time to save power. It is also designed to turn the robot off if it gets stuck against a wall for example, so it can be picked up and replaced. If the threshold for this count has been reached, the wheels are stopped and the 'status' variable is switched to 0, indicating that the robot should be powered off. The other special case is that the robot is stationary but there is at least one recognisable object currently in view. In this case the robot starts to move forward, and the stationary variable is set to false. This part of the else-if construct covering the special cases can be seen below in Figure 21.

```
if len(detections) == 0:
    if stationary:
        non_detection_count += 1
        if non_detection_count >= 10:
            status = 0
            wheels.stop()
        elif not stationary:
            non_detection_count += 1
            wheels.forward(0.8)
    elif stationary:
        non_detection_count += 1
        wheels.forward(0.8)
```

*Figure 21: Dealing with special cases for robot movement*

The remainder of the algorithm is the remainder of the else-if construct started with the special cases described above. The next direction of movement is determined by the `closest_object`'s physical position on the camera's frame, and its proximity to the robot as estimated by both the co-difference and the value of the 'distance' variable obtained from performing the ultrasonic pulse earlier. The physical position is determined by the location of the start of the object's bounding box. If the object is in close proximity to the robot, it will move in the opposite direction from where the object is. If the object is judged to be slightly further away then the robot will stay on its current movement path by continuing forward until it becomes closer to an object. A threshold of a metre away according to the ultrasonic sensor was decided on as the bar for an immediate change in direction by the robot. This algorithm allows a variety of decisions to be made by the robot, while remaining simple and compact enough to allow the cycle of image scanning and decision-making to continue as



quickly as possible. This is essential, as the state of the robot's view must be updated as often as possible in order to gain the best possible results, and allow it to effectively avoid all obstacles.

## **5. Evaluation**

### **5.1 Comparison With Similar Project**

To compare the design and work carried out on this project to a similar project, a paper that focusses on the process of creating a free-moving floor-cleaning robot from start to end<sub>[4]</sub>. It goes through the components used in turn, describing in detail why they were chosen and how they are utilised within the robot. It also details the tests that were carried out on the robot. This robot also uses two 2 dc gear motors in order to power its wheels, which is a similarity to this project, and most small robotics projects that you will find. In terms of what dictates the movement of the robot, the robot in the paper relies fully on data from ultrasonic sensors, whereas this project is more reliant on the picamera. Just using ultrasonic sensors is a valid way of doing this, but this project used the camera to give a more detailed view to the robot, that allows it to recognise specific objects and make more intelligent movement decisions based on this.

When it came to testing the robot, in the paper, straight line movement was the first consideration. *"The Odometry errors are the most important problem in a robot that has to follow a long path. In this first test, the robot had to follow a straight-line path for 8 m using only the encoders of the motors in the feedback control loop."* After this, the robot was tested in a 4x4 circuit shape. These two tests are very similar to the movement tests that were carried out in this project, as they cover the most basic and important factors when it comes to robot movement. The final test carried out in the paper is concerned with object recognition. *"Finally, for a small mobile robot, it is very important to have good detection skills for collision avoidance. In this test, several objects were placed at different locations in a plane in front of the robot. For this experiment, it is more important to know if the objects are detected rather than the measurement accuracy."* This matches the testing principles for this project. It is optimal if the measurements taken by the sensors and calculated from the camera stills are as accurate as possible, but the most essential thing is that the ability to correctly recognise

that obstacles in the view are there, and be able to avoid them. In this final test in the paper, the results were that objects “*are always detected at least 20 cm before collision, enough distance to stop or maneuver the robot.*” When performing early testing on the this project, this was not always the case. The robot would sometimes not pick up objects quickly enough and run into things. This could suggest that using a multitude of ultrasonic sensors, placed all around the body of the robot, could be more effective than using just one ultrasonic sensor as well as the camera. The camera itself could also be a possible factor. While I was able to verify that the picamera was capable of recognising objects in the individual component testing stage, it could be that on the move it has more trouble keeping track of objects. However, it is also worth keeping in mind that this paper was conducted by 4 highly-skilled academics whose knowledge and ability would be expected to be more of that than a fourth-year university student. In saying that, it is clear that both of these projects were designed with similar goals in mind, keeping to the same basic principles of robotics throughout, despite being intended for very different purposes.

## **5.2 Testing for Evaluation**

This section lines out the process of testing the full robot to evaluate its ability to perform the role that it was designed to do. An issue with testing the robot for evaluation is that its intended purpose is to be used to entertain cats. As I did not have any access to a cat to test the project on, any testing done has to attempt to simulate the situations that the robot would be used in, instead of being able to see real-time results of the robot interacting with a cat. This calls into question how much about the robot’s ability to actually carry out its intended purpose can be found out here. However, without having the physical presence of a cat, there are still several ways in which you can test the robot’s ability. These tests will still be able to give a good scope of the robot’s capabilities as a free-moving AI product. The first of these is to lay out an area with objects that restricts the robots’ movement to that certain area by placing it inside of it. The way that this was designed for this project, is by creating a circle with books, so that there are no gaps in which the robot could get through. A success in this test will be that the robot moves so that it remains within the circle, and it does not run into the edge of the circle or hit any of the books

by attempting to leave that area. When the robot passes this test, this can then be advanced by placing other objects inside the circle, with the robot having to stay within the circle in addition to avoiding these extra obstacles in order to pass the test. The next stage of testing the robot's ability, is to clear the floor of a room of any movable objects, and allow the robot to operate within that room. To be successful here, the robot should be able to move to all areas of the room that are not impeded with objects such as chairs or tables, and it should not run into any of these objects or any of the boundary walls of the room. To test its ability to operate in all areas of the room, the robot should be picked up and placed in different points in the room to start, including points closest to walls and other obstacles, as well as points of the room that have the most free space for the robot to move in. Similarly to the first test, this can be advanced once the robot passes by placing various objects in areas of the room where the floor space was previously free of obstacles. These tests should be performed in various environments where possible and in different lighting conditions, so that the ability of the robot to move throughout different environments effectively has been tested thoroughly.

The other main thing to test is the robot's reaction to moving objects, which should simulate its reaction to the main focus of its attention, the cat. The objective of the cat ideally, is for it to chase the robot, therefore moving towards it. This kind of movement can be simulated by holding an object in hand, and moving it towards the robot at varying speeds and directions. To be successful in this test, the robot should always move away from the object that is being moved towards it, except in the case that that this would involve the robot driving directly into another obstacle. Another way of simulating the robot's response to moving objects is to stand in the room the robot is operating in and walk towards it, the robot should attempt to move away from the person walking towards it, and if they get close enough it should attempt to move in the opposite direction completely.

Unfortunately, when it came to testing for evaluation, even through the tests that the robot should be put through had been designed previously, these could not be carried out in time for the submission of this report due to a fault in the Raspberry Pi. The Pi at some point, stopped taking power in from devices that it had been previously.

When the Pi is powered up, it indicates this through a green flashing light. When it became time to carry out these tests, that was not happening. This was true both when the power pack was plugged into the Raspberry Pi as it would be when the robot was operational, and when the Pi was connected through USB to the laptop. I attempted using multiple different USB cables to rule out the possibility of the cable breaking. Each cable used had the same result. The fact that the Pi would not power up by either USB or power pack rules out an issue with the power pack. This indicates that the issue is with the Raspberry Pi itself, requiring new hardware to get things running again, which unfortunately could not come and be set up in time for this report to be handed in.

### **5.3 Review of Project Goals**

This section will look back at the project goals that were laid out at the beginning of this report. Taking each in turn, it will attempt to determine if, or to what extent, this project lived up to each of these goals, looking at the less important stretch goal and minor goals first. The three major goals set at the start of the project will then be considered, as these are the most important goals for the project to have met. The work done to meet these major goals, and to what extent that they have been achieved, will give the best indication of how well the project has gone as a whole.

With regards to the stretch goal for the project, which was 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, this was not achieved. The stretch goal was outlined as a best-case scenario, something that could be brought into the project if the rest of it ran completely smoothly with minimal issues. Since there were a number of issues experienced throughout the process, the stretch goal had to be forgotten in preference of working towards the major goals of the project.

The first minor goal of the project was 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. The movements the robot was able to make was advanced by the use of the OpenCV deep learning object recognition framework, and incorporating this into the project allowed the robot to be able to recognise particular objects in its field of view, as stated in the goal. Despite this being incorporated into the software, the verification and testing of this that could be carried out was limited due to the hardware issues described earlier in the evaluation section. The other minor goal was to create an aesthetically pleasing robot to the cat to ensure the highest possible chance for the cat to be engaged with the robot. This was done by selecting a brightly coloured structure for the robot, as well as placing the components that protrude from the structure in such a way that they give the robot a sense of character.

The first major goal is to have the robot performing basic movement on its own across surfaces in all possible directions. The motors and wheels are utilised using the gpiozero Python library in order to achieve movement. To aid movement across surfaces, the placement of the wheels and the ball castor was vital to make sure friction does not hinder or prevent the movement of the robot. Another major goal was to implement basic AI object recognition. This was done by testing and verifying that the picamera was capable of performing object recognition, and including this in the robot by utilising the picamera python library. The final major goal was to house all components as compactly as possible within the robot to provide protection to the hardware, and because the robot itself has to be a reasonable size be durable and structurally sound. This was done by reusing a structure initially used to store something else, and adapting it to be used for this project. It was tested to make sure it could handle a reasonable amount of stress and pressure in order to protect the components housed inside.

## **6. Conclusion**

In conclusion, this project aimed to create an intelligent robot that provides entertainment for cats, built using principles of robotics and AI. It did this through creating a robot with the intention of having a cat chase it down, by using object

recognition technology to make its main focus avoiding this target as well as any other obstacles in its path.

## 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>
- [4] J. Palacin, J. A. Salse, I. Valganon and X. Clua, "Building a mobile robot for a floor-cleaning operation in domestic environments," in *IEEE Transactions on Instrumentation and Measurement*, vol. 53, no. 5, pp. 1418-1424, Oct. 2004, doi: 10.1109/TIM.2004.834093.