





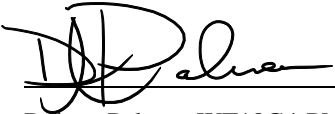
 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 1 of 26 Date: 29 October 2018
--	---	--

# WildTracker

## Final System Design


  

  
Prepared by \_\_\_\_\_ Date 29/10/2018  
Alex Santander, Tim Vu & Christian Neilsen,


WT18G4-GUI-OD  

  
Checked by \_\_\_\_\_ Date 29/10/2018  
Delenn Palmer, Christian Neilsen

WT18G4-PM-OD  
  
Approved by \_\_\_\_\_ Date 29/10/2018  
Delenn Palmer, WT18G4-PM

Authorised for use by \_\_\_\_\_ Date 29/10/2018  
Felipe Gonzalez, Supervisor


Queensland University of Technology  
Gardens Point Campus  
Brisbane, Australia, 4001.

This document is Copyright 2018 by the QUT. The content of this document, except that information which is in the public domain, is the proprietary property of the QUT and shall not be disclosed or reproduced in part or in whole other than for the purpose for which it has been prepared without the express permission of the QUT

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 2 of 26 Date: 29 October 2018
--	---	--


### Revision Record

<b>Document Issue/Revision Status</b>	<b>Description of Change</b>	<b>Date</b>	<b>Approved</b>
1.0	Final Issue	29/10/18	Deleenn Palmer

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 3 of 26 Date: 29 October 2018
--	---	--


## Table of Contents

Paragraph	Page No.
1 Introduction .....	5
1.1 Scope .....	5
2 Reference Documents.....	6
2.1 QUT Systems Engineering Documents.....	6
2.2 Numbering Scheme .....	6
3 Systems Introduction .....	7
4 System Architecture .....	8
4.1 Subsystem Architecture – Graphical User Interface .....	8
4.2 Subsystem Architecture – Object Detection .....	10
4.2.1 DarkFlow OD System .....	10
4.2.2 Initial Training.....	11
4.2.3 Custom Dataset Training.....	11
4.2.4 GUI Integration .....	12
4.2.5 JSON data display .....	12
5 System Design.....	13
5.1 Subsystem Design – Graphical User Interface.....	13
5.1.1 Subsystem Requirements .....	13
5.1.2 Software Choice .....	13
5.2 Subsystem Design – Object Detection .....	14
5.2.1 Subsystem Requirements .....	14
5.2.2 Software Choice .....	14
6 Conclusion.....	16
7 Appendices .....	17
7.1 Pseudo Code – Graphical User Interface .....	17
7.2 Pseudo Code – Object Detections .....	<b>Error! Bookmark not defined.</b>

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 4 of 26 Date: 29 October 2018
--	---	--

## Definitions

WildTracker (WT)	Software implementation of an autonomous detection and tracking tool for wildlife
QUT	Queensland University of Technology
HLO	High Level Objectives
PMP	Project Management Plan
GUI	Graphical User Interface

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 5 of 26 Date: 29 October 2018
--	---	--

## **1 Introduction**


The Final Systems Design document contains the finalised designs for the WildTracker application. This document is considered to be the final step of the systems engineering approach. This final phase outlines the developments made to each subsystem after the original design choices made in the ‘Preliminary Designs’ and testing that was completed to ensure that each subsystem met the system requirements outlined in ‘RD/2’. This document is the final iteration of the system design choice and contains all the finalised features of both the Graphical User Interface Subsystem and Object Detection Subsystem.

### **1.1 Scope**

This design document outlines all of the relevant factors that each subsystem required for complete functionality of the WildTracker application. It introduces both the GUI & OD Subsystems and how their intended features will fulfil the system requirements outlined. The document includes system architectures that provides the idea of the software flow throughout the GUI & OD subsystems both individually and collectively. Evidence supporting the justification for the design choices for each subsystem are also provided where appropriate.

### **1.2 Background**

The Queensland University of Technology (QUT) have appointed Group 4 of the EGH455 (Advanced Systems Design) class to design a software focused application that is capable of processing drone video footage and detecting specific animals in that footage with a high level of accuracy. Machine Learning (ML) and Deep Learning (DL) techniques will be employed to build classifiers which correctly identify the given animals in footage, and these classifiers are to be connected to a simple GUI which allows a user to load, display and process a video file. In addition to this, the classification results/statistics are to be shown on screen, and a final video file showing the detected animals should be saved to the user’s device for further ground truth analysis. More information regarding this project can be found in the Project Brief.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 6 of 26 Date: 29 October 2018
--	---	--

## 2 Reference Documents

### 2.1 QUT Systems Engineering Documents

RD/1	WT18G4-SUP-Customer Needs	Autonomous detection and tracking tool for wildlife
RD/2	WT18G4-SR-01	WildTracker Project: System Requirements Document 2018
RD/3	WT18G4-PMP-04	WildTracker Project: Project Management Plan 2018
RD/4	WT18G4-ICD-01	WildTracker Project: Interface Control Document 2018
RD/5	WT18G4-GUI-TR-01	WildTracker Project: Graphical User Interface Testing Report 2018
RD/6	WT18G4-OD-TR-01	WildTracker Project: Object Detection Testing Report 2018


### 2.2 Numbering Scheme

For ease of identification, a numbering system has been developed.

For the requirement REQ-M-01:

REQ – This is a requirement derived from the client's brief and the associated HLOs

M – Denotes a *mandatory* requirement, whereas D denotes the *desired* requirement.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 7 of 26 Date: 29 October 2018
--	---	--

### 3 Systems Introduction

The WildTracker system was designed to allow a user to perform object detection (OD) on raw drone footage through the use of a simple and intuitive graphical user interface (GUI). The system processes wildlife footage from top-down or oblique angles and using OD and machine learning (ML) techniques, the program will detect and identify animals in each frame of the raw input footage. The two major subsystems that make up WildTracker application are the GUI and the OD subsystem.

The GUI must allow a user to load, play, and analyse the video, as well as display relevant information in an enclosed pane. This must all be done in accordance with the System Requirements (REQ-M-05, REQ-M-06, REQ-M-07). Without the GUI, the output from the machine learning video analysis would not be able to be displayed and the project would not be able to meet the customer requirements. This makes the GUI subsystem critical to the WildTracker project.

The OD subsystem provides all of the backend video processing and OD functionality required to meet all of the functional and performance requirements set out by the customer (REQ-M-01, REQ-M-02, REQ-M-03, REQ-M-04). The DarkFlow system was used with the cutting-edge YOLOv2 OD architecture in order to train a ML model capable of accurately detecting the trained classes (i.e. elephants and horses) in a wide range of settings. Exported video files are created through the OD process, where the raw drone video file is looped through, with OD being performed on each individual frame, and with the resulting labels, confidence values and bounding boxes being displayed and exported to a new video file using OpenCV.

Through the successful integration of the GUI and OD subsystems all of the customers' requirements were met.

## 4 System Architecture

### 4.1 Subsystem Architecture – Graphical User Interface

#### 4.1.1 GUI Subsystem Architecture Diagram

The following diagram shows the architecture diagram for the GUI subsystem:

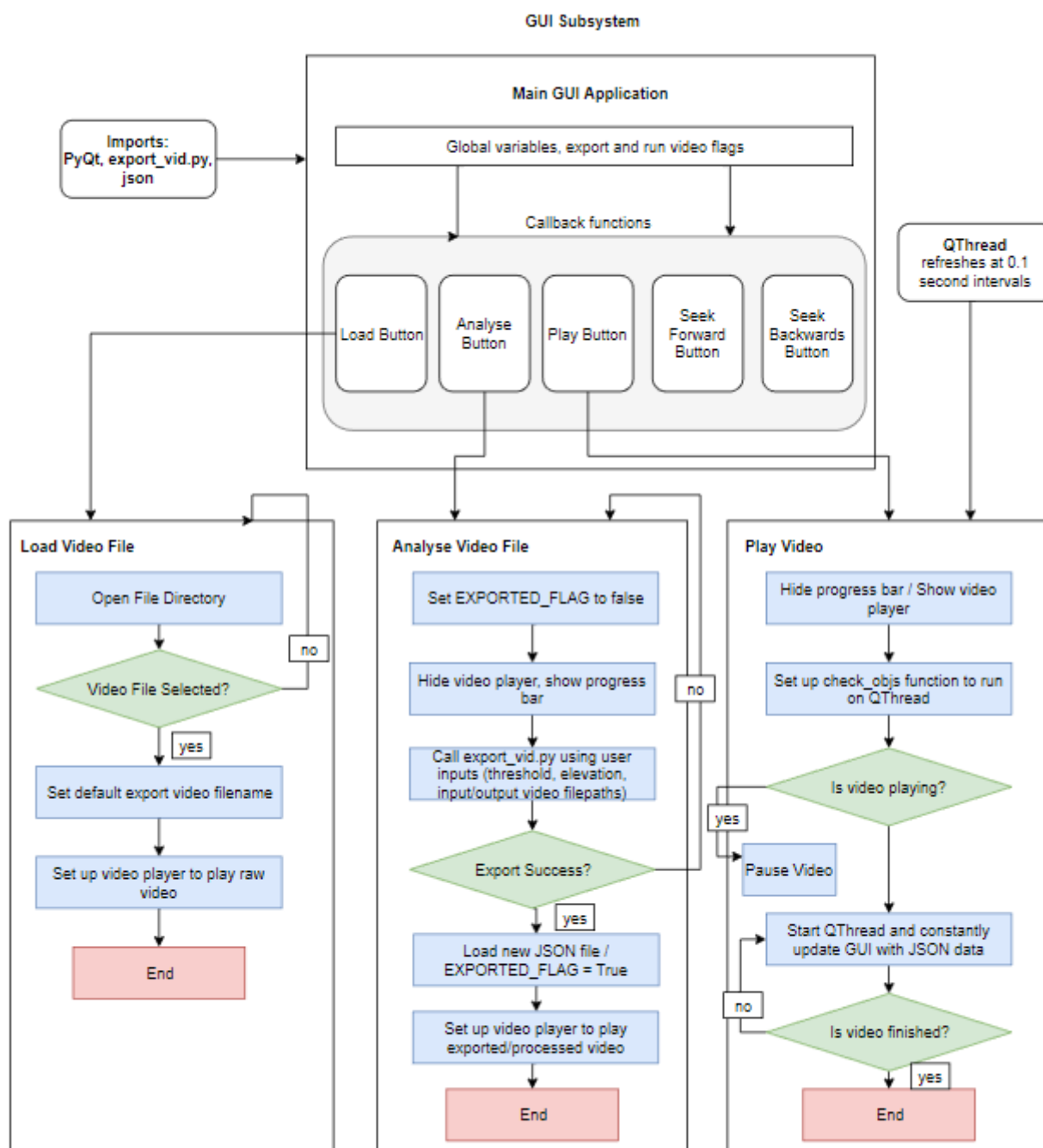



Figure 1 - GUI Subsystem Architecture




 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 9 of 26 Date: 29 October 2018
--	---	--

#### 4.1.2 GUI Subsystem Description

Although there was little variety in the method to implement the GUI, the one that was highlighted, PyQt GUI Framework alongside QtCreator, proved to be the most sufficient to implement what was required of the application. PyQt's framework provided the necessary base upon which the GUI could be built.

Before active development began on the GUI, it was initially planned to include two video players within the GUI, one for viewing the machine-learning-processed video and one to play the original. Additionally, a seek bar was planned to be added for ease of navigation in long videos. However, upon review of the system requirements it was determined that the two video players were unnecessary and that the additional video player to play the original video only served to clutter the GUI. The seek bar was also not included due to the revelation that the datasets which would be analysed were going to be short and would not make use of the seek bar. With the removal of these features the motivation for the GUI was clear: the GUI was to be as minimalistic as possible to reduce clutter and allow for ease of use.

With the idea for the GUI set, work began on the design and creation of the GUI in PyQt. The first element of the GUI that was implemented was the video player and its accompanying controls. This involved creating a custom video player widget as there was no built-in video player widget in QtCreator. Once this was completed, buttons to seek forward, backward, pause and play the video were added. With the GUI element for the video player complete, buttons to import the original video and analyse this video were added. Within the code to analyse the video, the Object Detection python programming was included. Alongside these buttons were two textbox elements which display the path to the original video selected and the processed video. The last part, critical to the GUI, was the implementation of a label to display the number of animals in the video frame and a list box which would display a list of the animals and their sizes. The list box and label contains information that is fed from the Object Detection subsystem. Information from the Object Detection subsystem was also used to implement a progress bar that would display the progress that was being made on the video analysis.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 10 of 26 Date: 29 October 2018
--	---	---

## 4.2 Subsystem Architecture – Object Detection

### 4.2.1 OD Subsystem Architecture Diagram

The following figure shows the architecture diagram for the OD subsystem:

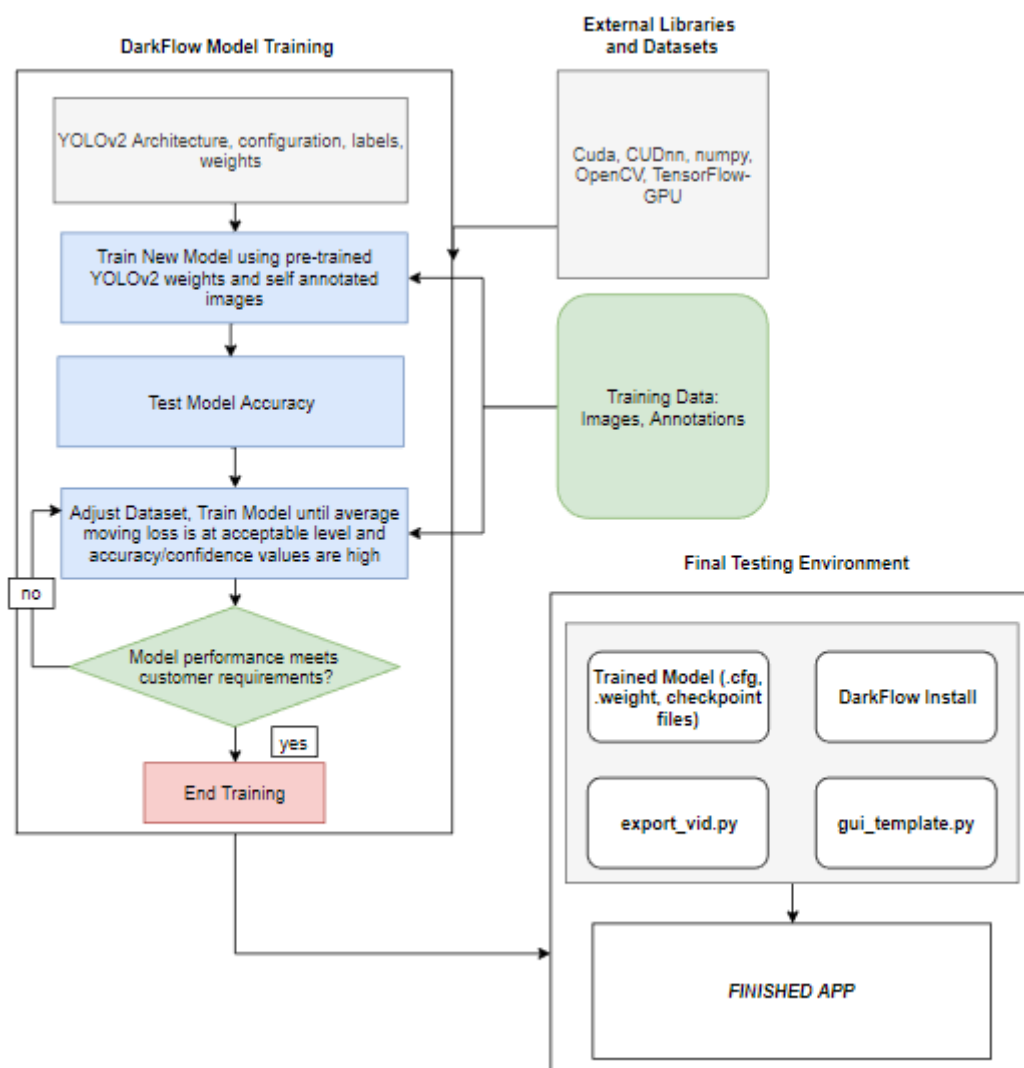



Figure 2 - OD Subsystem Architecture Diagram

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 11 of 26 Date: 29 October 2018
--	---	---

## **4.2.2 OD Subsystem Description**

### **4.2.2.1 DarkFlow OD System**

In the initial stages of this project many avenues were researched and presented to the client regarding how to implement machine learning to achieve the desired outcome. After a period of research, a choice had to be made between the DarkFlow and Keras OD methods. Each strategy had its own merits but ultimately, the widely acclaimed Darkflow system along with its pre-trained weights was decided upon. The two key reasons for using DarkFlow over Keras were:


- DarkFlow provides easy model testing and training through the command line.
- DarkFlow allows for an easier and more portable installation process.

### **4.2.2.2 Initial Training**

Two stages of initial training were performed before the final model design was chosen. Initially, the default pre-trained YOLOv2 weights and configuration based on the COCO dataset was used. The major problem with using this dataset was the lack of relevant images which the model was trained on. Although there were several thousand annotated images for each of the desired classes in the COCO dataset, most of the images did not feature shots from elevation (top-down or oblique angle), but instead primarily consisted of ground level images of each animal. Because of this, the accuracy of the default model was extremely low when tested on drone footage, and so an entirely different approach was taken.

### **4.2.2.3 Custom Dataset Training**

A script was written (see section 7 Appendices) which simply extracted frames from videos. Knowing that the format of the client's video would be a bird's eye view with medium elevation, videos with similar footage were collated, extracted from and then manually annotated (see 5.2.2 software choice). For the process of training the model the default YOLOv2 weight and configuration files were used as a base, and the custom database of images and corresponding annotations were used to train on-top of this base. For details regarding the setup of the training environment please refer to WT18G4-OD-TR.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 12 of 26 Date: 29 October 2018
--	---	---


#### **4.2.2.4 GUI Integration**

In the early stages of development variables that would later be fed through by the GUI (see section 4.1 above) would hard coded in, such as the filename, destination and threshold values. From here the trained Darkflow model was imported and various video reading and writing objects were made using OpenCV. Options for setting the exported video codec, framerate and resolution were also set here.

From here the raw video frames were iterated through, threading each one within the Darkflow model, thereby detecting the relevant objects in each frame. Size estimations were made for each detected object based on the label of the predicted result. For added flexibility, elevation and threshold variables were created in the export\_vid.py function, so that these values could be set and changed through the GUI by a user to fine tune the model for better prediction and size estimation.

#### **4.2.2.5 JSON data display**

It was important that the GUI was able to display the relevant detection data for each frame of a processed video file. In order to achieve this while maintaining responsiveness of the GPU, a JSON file was created and all relevant frame data was stored in a Python object, and eventually exported to this file. When a user chose to play an exported video file using the application, the data for the particular frame being shown is grabbed from the JSON file and the total number of animals, labels and size estimates is displayed and updated as the video plays back. The GUI was kept responsive by using QtCore Threading.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 13 of 26 Date: 29 October 2018
--	---	---

## **5 System Design**

### **5.1 Subsystem Design – Graphical User Interface**

#### **5.1.1 Subsystem Requirements**

GUI design is not particularly intensive on its own. It does not require any specialist hardware and can be performed on most modern PCs and laptops. In this project the GUI was mostly designed on a low-end laptop without any dedicated graphics.

#### **5.1.2 Software Choice**

##### **5.1.2.1 Python**


The primary choice of software language for the GUI subsystem was Python. Python is a high-level, easy to deploy and readable dynamic programming language that allows multiple programming paradigms to be used. Some of the main advantages of using python are the relative ease of use when compared to other Object Orientated (OO) programming languages such as C++, Java, and also its huge popularity. Thanks to its popularity, a large number of frequently used and well documented frameworks were available for GUI design, with the PyQt framework being one of those. As of writing this the current version of Python being used is v3.6.

##### **5.1.2.2 PyQt and QtCreator**

The PyQt framework and QtCreator were used alongside Python to develop the GUI. PyQt is a GUI framework that provides an API to easily access GUI elements from within Python. QtCreator provided a user interface for PyQt which allows for placing the GUI elements in a .ui file which would later be converted to PyQt API calls in a .py file.

##### **5.1.2.3 Anaconda**

Anaconda provides a free an open source Python distribution, and through the Anaconda Prompt allows users to easily manage installed packages and user environments. By using simple command line inputs, a user can install and manage all necessary packages to run the application on their own.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 14 of 26 Date: 29 October 2018
--	---	---

## **5.2 Subsystem Design – Object Detection**

### **5.2.1 Subsystem Requirements**

Machine Learning (ML) is the process of training an algorithm to generate models which can apply their newly learned classification properties to other data. However, none of this can be achieved without the necessary hardware. TensorFlow has designed their software for optimal efficiency with Nvidia GPU's, two of our members were in possession of a high-end model which was ideal for training our algorithm.

### **5.2.2 Software Choice**

#### **5.2.2.1 Python**

The primary choice of software language for the OD subsystem was Python. Python is a high-level, easy to deploy and readable dynamic programming language that allows multiple programming paradigms to be used. Some of the main advantages of using python are the relative ease of use when compared to other Object Orientated (OO) programming languages such as C++, Java, and also its huge popularity. Additionally, Python was found to be the best way to streamline integration because all systems were collectively chosen to be written in it. As of writing this the current version of Python being used is v3.6.

#### **5.2.2.2 OpenCV and TensorFlow**


The TensorFlow and OpenCV libraries were used alongside Python to develop the OD subsystem. The Python OpenCV (Open Computer Vision) library is an open source library developed by intel, allowing for the pre and post processing of drone footage in conjunction with the NN classifiers provided by TensorFlow.

#### **5.2.2.3 Anaconda**

Anaconda provides a free an open source Python distribution, and through the Anaconda Prompt allows users to easily manage installed packages and user environments. By using simple command line inputs, a user can install and manage all necessary packages to run the application on their own.


#### **5.2.2.4 Darkflow**

Darkflow is an out of the box tool used OD by providing a Python wrapper for the Darknet NN framework and making use of the YOLOv2 architecture. The model comes with pretrained weights for either the COCO dataset or VOC PASCAL. With very simple installation and execution it is a very user friendly and effective modern tool for object detection.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 15 of 26 Date: 29 October 2018
--	---	---

#### **5.2.2.5 LabelImg**


A simple desktop tool with the purpose of loading images within its interface. The user manually clicks and drags to create bounding boxes with labels to specify what and where the model is training within the image. The annotations are stored in an .xml file to be read in conjunction with the original image by the model.

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 16 of 26 Date: 29 October 2018
--	---	---

## 6 Conclusion

By integrating the separate GUI and OD subsystems, the final WildTracker system was successfully created. By using DarkFlow, YOLOv2, OpenCV and a custom annotated dataset, the OD subsystem was able to meet all of the functional and performance requirements of the system (REQ-M-1,2,3,4). Alongside the OD subsystem was the GUI subsystem which enabled us to meet requirements REQ-M-5,6,7. By following sound engineering design principles and processes, a final product that meets all of the customers' requirements has been created.



 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 17 of 26 Date: 29 October 2018
--	---	---

## 7 Appendices

### 7.1 Graphical User Interface Code Snippets

The following code snippet was used to create the GUI visually and implement functions to load, analyse and export the video after being processed.

#### gui\_template.py code

```
from PyQt5 import QtWidgets
from PyQt5 import QtCore, QtMultimedia
from PyQt5.QtWidgets import QFileDialog, QProgressBar
from example_window import Ui_WildTracker
import sys, os, re, tempfile, threading, time, json, traceback
import export_vid

# Creating global flag to check if video export finished
finished_exporting = False
filename = ''
filename_export = ''
url = ''
url_export = ''
thresh = 0
elevation = 0
json_data = ''
RUN_FLAG = True
EXPORTED_FLAG = False

class QThread1(QtCore.QThread):

    sig = QtCore.pyqtSignal()

    def __init__(self, parent=None):
        QtCore.QThread.__init__(self, parent)

    def run(self):
        while RUN_FLAG:
            self.sig.emit()
            time.sleep(0.0001)

# The class that handles the application itself
class ApplicationWindow(QtWidgets.QMainWindow):
    def __init__(self):
        # Create the Qt5 application backend
        super(ApplicationWindow, self).__init__()

        # Load in and display the UI
        self.ui = Ui_WildTracker()
        self.ui.setupUi(self)
```



```
#Configure the video widgets
self.video_player = QtMultimedia.QMediaPlayer(None,
QtMultimedia.QMediaPlayer.VideoSurface)

#Connect events
self.ui.button_1.clicked.connect(self.callback_button_1)
self.ui.button_2.clicked.connect(self.callback_button_2)
self.ui.button_3.clicked.connect(self.callback_button_3)
self.ui.button_4.clicked.connect(self.callback_button_4)
self.ui.button_5.clicked.connect(self.callback_button_5)

#Load video button
def callback_button_1(self):
    print("Button 1 pressed!")

    global finished_exporting, filename, filename_export, url,
url_export

    # Grabbing filename using windows explorer like GUI box
    file = QFileDialog.getOpenFileName()
    url = QtCore.QUrl.fromLocalFile(file[0])
    self.ui.textEdit.setText(file[0])
    filename = self.ui.textEdit.toPlainText()

    # Checking if a file was chosen (valid file names) and setting
default export video name
    if len(filename) > 0:
        filename_export = os.path.splitext(filename)[0] +
'_exported.mp4'
    else: filename_export = ''

    url_export = QtCore.QUrl.fromUserInput(filename_export)

    self.ui.textBrowser.setText(filename_export)
    finished_exporting = False

    # Set up the video output widget to play raw video
    self.video_player.setVideoOutput(self.ui.video_widget)
    if not finished_exporting:

self.video_player.setMedia(QtMultimedia.QMediaContent(url))

#Analyse video button
def callback_button_2(self):
    print("Button 2 pressed!")
```

```
global finished_exporting, filename, filename_export, url,
url_export, thresh, elevation, json_data, EXPORTED_FLAG

# Set flag to false to prevent empty JSON file data access later
EXPORTED_FLAG = False
self.ui.video_widget.hide()

filename = self.ui.textEdit.toPlainText()
filename_export = self.ui.textBrowser.toPlainText()

self.video_player = QtMultimedia.QMediaPlayer(None,
QtMultimedia.QMediaPlayer.VideoSurface)

# Visually show video processing, predict bounding boxes and
export video

#Process using machine learning side
#Video is exported to directory

# Setting GUI and initial values for exporting
self.ui.progress.show()
self.ui.progress_label.show()
progress_val = 0.00
thresh = float(self.ui.textThresh.toPlainText())
elevation = float(self.ui.textElevation.toPlainText())

# Main external function used to perform object detection
(references export_vid.py)
export_vid.process(progress_val, self.ui.progress,
self.ui.textBrowser_3, filename, filename_export, thresh, elevation)
self.ui.progress.setValue(100)
finished_exporting = True

# Globally loading JSON data when export finishes
global json_data
with open('tester.json') as f:
    json_data = json.load(f)

# Set up the video output widget
self.video_player.setVideoOutput(self.ui.video_widget)
if finished_exporting:

self.video_player.setMedia(QtMultimedia.QMediaContent(url_export))

# Set flag to true now JSON file can be fully read
EXPORTED_FLAG = True

#Play button
def callback_button_3(self):
```



```
print("Button 3 pressed!")

self.ui.progress.hide()
self.ui.progress_label.hide()
if self.ui.video_widget.isHidden(): self.ui.video_widget.show()

# Call to PyQt thread (multithreading needed to seamlessly display
detection data for exported video)
self.thread1 = QThread1()

global json_data, RUN_FLAG, EXPORTED_FLAG

# Primary display function that grabs from generated JSON file and
displays
# object detection metrics (labels, number of detections, size
etc.)
def check_objs():
    if EXPORTED_FLAG:
        total_time = json_data['total_length']
        current_time = self.video_player.position() / 1000
        time_ratio = (current_time / total_time) * 2
        round_frame = (int(round_down(time_ratio *
json_data['total_frames'], 1)))
        for frame in json_data['frame']:
            if frame['frame_count'] == round_frame:
                self.ui.listWidget.clear()

self.ui.textBrowser_3.setText(str(frame['num_objects']))
                for obj in frame['objects_type']:

self.ui.listWidget.addItem(str(obj))

    if self.video_player.state() ==
QtMultimedia.QMediaPlayer.PlayingState:
        RUN_FLAG = False
        self.video_player.pause()
    else:
        RUN_FLAG = True
        self.thread1.start()
        self.video_player.play()
        self.thread1.sig.connect(check_objs)


#Seek backwards 5 seconds
def callback_button_4(self):
    print("Button 4 pressed!")
    self.video_player.setPosition(self.video_player.position()-5000)
```

```
#Seek forward 5 seconds
def callback_button_5(self):
    print("Button 5 pressed!")
    self.video_player.setPosition(self.video_player.position()+5000)

# Used to get rounded frame count
def round_down(num, divisor):
    return num - (num%divisor)

# The "main()" function, like a C program
def main():
    print("Loading application...")
    app = QtWidgets.QApplication(sys.argv)
    application = ApplicationWindow()
    print("Application loaded.")
    application.show()
    sys.exit(app.exec_())

# Provides a start point for our code
if __name__ == "__main__":
    main()
```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 22 of 26 Date: 29 October 2018
--	---	---

## 7.2 Object Detection Code Snippets

### export\_vid.py code

The following code was used to perform OD using the trained DarkFlow model, and then to export the processed video to a new file (along with related .json data):

```
import cv2
from darkflow.net.build import TFNet
import numpy as np
import time
from tqdm import tqdm
from moviepy.video.io.VideoFileClip import VideoFileClip
import json

def process(progress_val, progress, animalCount, filename, filename_export,
thresh, elevation):

    # Load in the chosen darkflow model, weights, and other parameters
    option = {
        'model': 'cfg/yolov2-2c.cfg',
        'load': 132500,
        'threshold': thresh,
        'gpu': 0.8
    }

    # Construct TensorFlow Net using darkflow model parameters
    tfnet = TFNet(option)

    # Set raw and export video paths
    video_in = filename
    video_out = filename_export

    # Clip (used by moviepy to extract video duration parameter)
    clip = VideoFileClip(video_in)

    # Create video reader object for looping through raw video frames
    video_reader = cv2.VideoCapture(video_in)

    # Get necessary video parameters
    nb_frames = int(video_reader.get(cv2.CAP_PROP_FRAME_COUNT))
    frame_h = int(video_reader.get(cv2.CAP_PROP_FRAME_HEIGHT))
    frame_w = int(video_reader.get(cv2.CAP_PROP_FRAME_WIDTH))

    # Create video writer object and set video codec, framerate, resolution
    video_writer = cv2.VideoWriter(video_out, cv2.VideoWriter_fourcc(*'XVID'),
50.0, (frame_w, frame_h))
```

```
# Initialize data object and parameters to be updated frame by frame
progress_val = 0.00
data = {}
data['frame'] = []
data['total_frames'] = nb_frames
data['total_length'] = clip.duration

# Begin loop through raw video frames and perform object detection on each
frame
for i in (range(nb_frames)):
    ret, frame = video_reader.read()

    # Calculate export progress and show in console and GUI
    progress_val = (i / nb_frames) * 100
    print('{0:.2f}'.format(progress_val))
    progress.setValue(progress_val)

    input_image = cv2.resize(frame, (416, 416))
    input_image = input_image / 255.
    input_image = input_image[:, :, ::-1]
    input_image = np.expand_dims(input_image, 0)

    # Perform detection on frame
    results = tfnet.return_predict(frame)

    object_list = []
    size_list = []

    # Annotate each object and get object data (labels, size, bounding box
    coordinates)
    # Also writes necessary data to JSON object
    for result in results:
        tl = (result['topleft']['x'], result['topleft']['y'])
        br = (result['bottomright']['x'], result['bottomright']['y'])
        label = result['label']
        conf = result['confidence']
        lp = abs(br[0] - tl[0]) * abs(br[1] - tl[1])

        if label == 'elephant':
            estimate_size, la_val = estimate_size_elephant(lp, elevation, 910)
            if estimate_size:
                labelSize = label + ' (large)'
                frame = cv2.rectangle(frame, tl, br, (0,0,255), 2)
            else:
                labelSize = label + ' (small)'
                frame = cv2.rectangle(frame, tl, br, (255,0,0), 2)
            object_list.append(labelSize)
            size_list.append(la_val)
```

```
        elif label == 'horse':
            estimate_size, la_val = estimate_size_horse(lp, elevation, 910)
            if estimate_size:
                labelSize = label + ' (large)'
                frame = cv2.rectangle(frame, tl, br, (0,0,255), 2)
            else:
                labelSize = label + ' (small)'
                frame = cv2.rectangle(frame, tl, br, (255,0,0), 2)
            object_list.append(labelSize)
            size_list.append(la_val)

        frame = cv2.putText(frame, "{0} {1:.2f}".format(label, conf), tl,
cv2.FONT_HERSHEY_COMPLEX, 0.5, (100, 0, 255), 2)

    data['frame'].append({'objects_type': object_list, 'frame_count':
i, 'num_objects': len(results), 'size_val': size_list})

    # Write individual detection frame to exported video
    video_writer.write(np.uint8(frame))

    # quit processing action
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

    video_reader.release()
    video_writer.release()


    # finally write JSON object to JSON file to be accessed by GUI
    with open('tester.json', 'w') as f:
        json.dump(data, f, indent=4, sort_keys=True)

# Size estimation functions (la value depends on screen resolution, highly
variable)

def estimate_size_elephant(lp, elevation, frame_w):
    gsd = (32 * elevation) / (24 * frame_w) * 100
    la = gsd * lp
    if la < 12000:
        isBig = False
    else: isBig = True
    return isBig, la

def estimate_size_horse(lp, elevation, frame_w):
    gsd = (32 * elevation) / (24 * frame_w) * 100
    la = gsd * lp
```



 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b> <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 25 of 26 Date: 29 October 2018
--	---	---

```

if la < 5000:
    isBig = False
else: isBig = True
return isBig, la

```

### im\_extract.py code

The following script was used to sample video files in order to generate images for annotating and training the model:

```


import sys
import argparse

import cv2
print(cv2.__version__)

def extractImages(pathIn, pathOut):
    count = 0
    vidcap = cv2.VideoCapture(pathIn)
    success,image = vidcap.read()
    success = True
    while success:
        vidcap.set(cv2.CAP_PROP_POS_MSEC,(count*1000))    # added this line
        success,image = vidcap.read()
        print ('Read a new frame: ', success)
        cv2.imwrite( pathOut + "\\brapp%d.jpg" % count, image)    # save frame as
        # JPEG file
        count = count + 1

if __name__=="__main__":
    print("aba")
    a = argparse.ArgumentParser()
    a.add_argument("--pathIn", help="path to video")
    a.add_argument("--pathOut", help="path to images")
    args = a.parse_args()
    print(args)
    extractImages(args.pathIn, args.pathOut)

```

 <b>Queensland University of Technology</b>	<b>QUT Systems Engineering</b>  <b>WT18G4</b>	Doc No: WT18G4-FD-01 Issue: 1 Page: 26 of 26 Date: 29 October 2018
--	---	---