Team 307 - Emergency Management Drone

Operation Manual

Members

Haley Barrett	hnb15@my.fsu.edu
Kody Koch	kmk14g@my.fsu.edu
Juan Patino	jpp13d@my.fsu.edu
Joshua Reid	jnr14@my.fsu.edu
Matthew Roberts	mjr15d@my.fsu.edu
Francisco Silva	frs14@my.fsu.edu

Sponsor

Mr. David F. Merrick

Faculty Advisor

Dr. Rodney Roberts

Instructor

Dr. Jerris Hooker

Reviewers

Dr. Omar Faruque
Dr. Simon Foo

Acknowledgements

Team 307 would like to give a special thanks to our sponsor, David Merrick, for providing the opportunity for us to be a part of this project as well as being an active supporter of the project's progression. We would also like to express our gratitude to our advisor, Rodney Roberts, for providing constant advice and support.

Team 307 would like to also thank Dr. Harvey, who provided advice and knowledge about the communication systems.

Team 307 would like to thank Henry Troutman, who proposed the idea of using LoRa communication, as well as helped design the PCB that was used to prototype it.

Lastly, we would like to thank our design instructor, Jerris Hooker, for always providing technical advice as well as aiding us through each step of the project.

Table of Contents

List of Figures

1 Executive Summary

The use of drones is very important to utilize in dangerous situations or inaccessible terrain. Our sponsor, the Florida State Emergency Management and Homeland Security (EMHS) currently uses drones that are capable of autonomously scanning an area but are not able to provide the users with live footage. The Florida State EMHS contacted Team 307 to create a drone that uses a computer to find targets in an emergency situation.

The customer wanted longer flight time and range to have a more effective search and rescue option. To increase the flight time, batteries with more energy and a more effective power system were used. A better communication system was also implemented in order to reach the desired range. The drone was designed to be more like an airplane instead of a helicopter so that the drone can travel through the air more efficiently. The improved battery life of the design is estimated to be 56 minutes at ideal conditions and 26 minutes at max power conditions. The range target of 1 km has been met and the maximum range reached during testing was 3 km.

A reliable and consistent object detection is necessary to achieve these goals. A neural network was used to identify and highlight targets in the line of sight of the drone. The neural network was re-trained using a python script based on TensorFlow to only identify persons from its point of view. The neural network model used was the YOLOv3, a model able to process and identify objects at a large range. Each time a person is identified, a picture is taken and saved in the drone's microprocessor. This picture is then sent to the base station with coordinates of the object identified.

In order to accomplish the sponsor's ideal flight time of 20-minutes, the mechanical structure of the drone consisted of a fixed wing design with 48" (4ft) of total lateral size. Each wing will be 18" (1.5ft) long. The onboard electronics will be powered by a three-cell LiPo battery. Furthermore, a lightweight EPS foam frame was constructed with a brushless airplane motor and two 7X5" propellers to increase the drone's efficiency. A belly-landing method was incorporated to minimize the weight of the drone. The drone is equipped with a radio control (RC) transceiver, which allows for manual landing and acts as a fail-safe during autonomous missions. The user will plan the autonomous mission and deploy the route to the drone using the Pixhawk.

The final product increases the capability to conduct reliable and effective search and rescue mission. This was done by removing manual processing, resulting in an increase of autonomy for the drone's structure. Finally, a capable computer vision was used for real-time image processing.

2 Components

2.1 Arm Assembly

(a) **Motor** - Cobra 3510/16 1200Kv
   - Propeller – Two Gemfan composite 7x5 propellers
   - Propeller nut, washer

(b) **Electronic Speed Control (ESC)** - Cobra 60A
   - Power wire connecting to main power bus
   - Signal wire connecting to autopilot

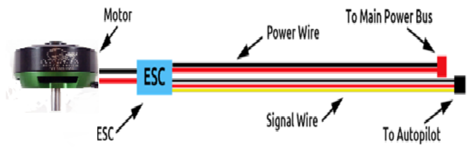


Figure 1: Electronic Speed Control diagram

Figure 1 shows the layout for the arm of the connections between the motor, the ESC, the main power bus, and the autopilot.

2.2 Fuselage Assembly

(a) **Autopilot** - Pixhawk PX4
- Power Meter (PM)
- GPS/Compass

(b) **11.1V to 5V Power Converter** - Power Converter PCB
- Converts voltage from 11.1V to 5V
- 85% efficiency

(c) **Main Power Bus**
- Distributes battery power among necessary components

(d) **Fuselage and Wings**
- 96”x48”x2” Sheet EPS Foam

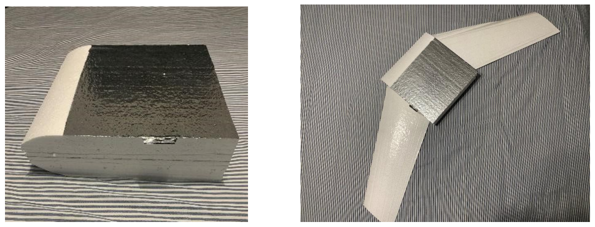
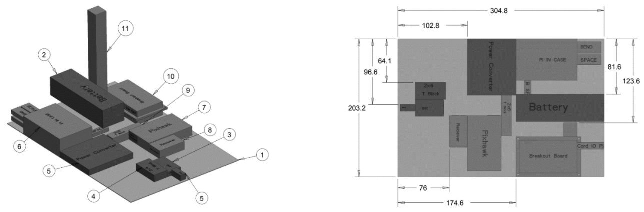

(a) Drone fuselage (b) Drone fuselage with wings

Figure 2: Drone Fuselage with and without wings

Figure 2 displays the drone’s fuselage and what it looks like with the drone’s wings

(e) **Battery** - 3 cell Lithium-Ion Polymer (LiPo)
- 8000mAh capacity

(f) **Telemetry Radio** - RFD900 Telemetry Radio
- Long distance telemetry
- Interfaces with Pixhawk



(a) 3D view (b) 2D view

Figure 3: Layout of all the electronics assembled inside the fuselage

In Figure 3, the left image shows all the components and their locations in the fuselage. Each component is labeled by part numbers. Part 1 is the base of the fuselage. Part 2 is the battery that will provide power to the drone. Part 3 is the ESC. Part 4 is the battery T block, and right next to it at part 5 is the motor protrusion. Part 6 consists of the main processor, the Raspberry Pi 3B+, which will be located inside a case. Part 7 is the Pixhawk. The Pixhawk oversees the autonomous route the drone takes. Part 8 is the telemetry radio that interfaces with the Pixhawk. Part 9 contains the 5V T block. Part 10 is the drone's breakout board. Finally, part 11 consists of the omnidirectional antenna the drone uses for the communication system.

2.3 RC controller, camera, microprocessor and wireless base station

(a) **RC Controller** - Taranis x9d
  - Connects with RFD Modems

(b) **Transceiver** - Adafruit RFM95W LoRa Radio
  - Radio transceiver module for long distance data communication

(c) **Camera** - 720p Logitech camera
  - Captures images of the desired objects and environment

(d) **Antennas** - Yagi antenna, Omnidirectional antenna
  - Yagi antenna offers high gain(10db) and heavy-duty construction
  - Signal wire connecting to autopilot

(e) **Microprocessor with attachments** - Raspberry Pi 3 B+, NCS 2 and Adafruit RFM95W
  - The Pi will handle the communication system and the image processing
  - The Neural Compute Stick 2 will help the Pi run the neural network in a higher frame rate (3FPS).
  - The Adafruit RFM95W interfaces with the Pi and handles the transfer of information between the drone and the base station.

3 Setup

3.1 Flight Controls

QGroundControl, a software used by the ground station to control the drone, is installed like any other regular program. The file can be found on the following website: qgroundcontrol.com. Once the program has been downloaded and installed, the user can connect to the drone through either USB or telemetry radio. For this project, two RFD900 modems were used. One modem connects to the Pixhawk, while the other modem connects to the computer being used at the ground station.

3.2 Raspberry Pi

The neural network is already up and running on the Raspberry Pi. There are four Github repositories that were used to train and run the neural network. These were darknet, Alexey darknet, Yolo Mark and OpenVINO-YoloV3. The Raspberry Pi have two modules running which are the image processing and the communication system.

3.3 Image Processing

To run the neural network on the Raspberry Pi, Open-Vino needs to be installed. This program manages the functionality of the Neural Compute Stick 2 (NCS2) which helps the processing of the neural network on the Pi. To install this program, the OpenVINO-YoloV3 repository was used . The first step is to clone this repository using the following command:

- git clone https://github.com/PINTO0309/OpenVINO-YoloV3.git

After cloning the repository, the following commands are used to install OpenVINO on the Raspberry Pi:

```
$ sudo apt update
$ sudo apt upgrade
$ curl -sc /tmp/cookie "<INSERT_LINK_HERE>"
```

Where the link is: `https://drive.google.com/uc?export=download&id=1NFt6g6Zkn eHioU2P7rUJ8BFpQhIazbym>/dev/null`

```
$ CODE = "$(awk␣'/_warning_/␣{print␣$NF}'␣/tmp/cookie)"
$ curl -Lb /tmp/cookie "<INSERT_LINK_HERE>" -o
> l_openvino_toolkit_raspbi_p_2019.1.133.tgz
```

Where the link is: `https://drive.google.com/uc?export=download&confirm=${CODE}&id=1NFt6g6ZkneHioU2P7rUJ8BF`

```
$ tar -zxvf l_openvino_toolkit_raspbi_p_2019.1.133.tgz
$ rm l_openvino_toolkit_raspbi_p_2019.1.133.tgz
$ sed -i "s|<INSTALLDIR>|$(pwd)/inference_engine_vpu_arm|"
> inference_engine_vpu_arm/bin/setupvars.sh
$ nano ~/.bashrc
```

In this step, add `/home/pi/inference_engine_vpu_arm/bin/setupvars.sh` at the end of the file and exit saving the file. After that, keep following the line of commands:

```
$ source ~/.bashrc
```

When this command is input you should see an output on the terminal that has [setupvars.sh] OpenVINO environment initialized. This will appear every time that a terminal is open which makes your OpenVINO environment work globally. The following set of commands update the USB configurations on the Pi to be able to use the NCS2 stick:

```
$ sudo usermod -a -G users "$(whoami)"
$ sudo reboot
```

After running all the commands, the Pi should have OpenVINO installed. The next step would be to download the Team 307 repository. This Github includes the code and the weights used to run the neural network. To clone, use the following command:

```
$ git clone https://github.com/frank010/Team_307_code.git
```

From this repository copy the weight files which are the .bin and .xml file to the path described below

```
$ /home/pi/OpenVINO-YoloV3/
```

The python script called `tiny_test.py` used to run the image detection should also be copied to the main folder of OpenVINO which path is described below:

```
$ /home/pi/OpenVINO-YoloV3/
```

3.4 Communication System

To run the communication system on both of the Raspberry Pi's we need to install the CircuitPython libraries and download the font5x8.bin file. These libraries include the functionality for the framebuf and the RFM9x module. The following code installs both of the libraries and the .bin file:

```
$ sudo pip3 install adafruit-circuitpython-framebuf
$ sudo pip3 install adafruit-circuitpython-rfm9x
$ git clone https://github.com/adafruit/Adafruit_CircuitPython_framebuf.git
```

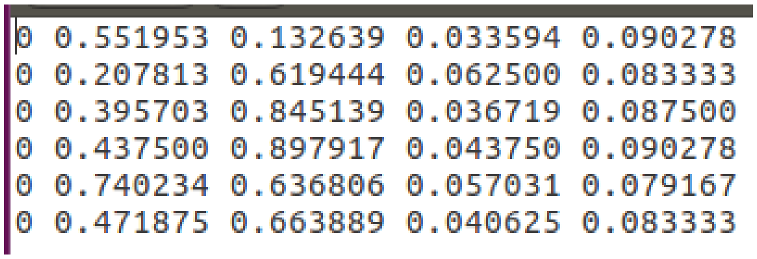
The font5x8.bin can be found the following path:

- `/home/pi/Adafruit_CircuitPython_framebuf/examples/`

The Tx and Rx code can be downloaded from the Team 307 Github. After all the files are downloaded and the libraries are installed, the Rx and Tx code should be moved to the desktop. The font5x8.bin must also be in the same directory as the code.

4 Training the Neural Network

To train the neural network, a Linux system is recommended with Ubuntu 16.04 installed and a powerful GPU. This process is very extensive and it has many steps, but the most important one is the dataset. The dataset should include at least 6,000 pictures. The number of positive images should be around 2,000 and negatives should be 4,000. The positive images should include pictures of single and multiple targets in the environment desired for object detection. The negative images should also represent the environment desired for detection, but they must not have any objects that need to be detected. The images should be very diverse for the training to be effective. After acquiring all the images for the dataset labeling is required for each of the images. Each image on the dataset must have a .txt file with the same name where the labeling is stored. For the darknet format the labeling should be the same as the one shown in the figure below:



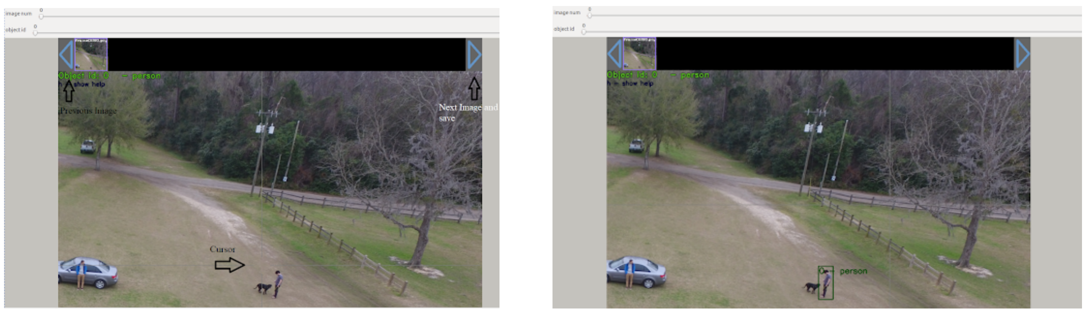
```
0 0.551953 0.132639 0.033594 0.090278
0 0.207813 0.619444 0.062500 0.083333
0 0.395703 0.845139 0.036719 0.087500
0 0.437500 0.897917 0.043750 0.090278
0 0.740234 0.636806 0.057031 0.079167
0 0.471875 0.663889 0.040625 0.083333
```

Figure 4: Multiple objects in label

Labeling should be done using the software created in the Yolo Mark Github. This application loads each of the images to user friendly interface where they can be labeled manually. The process to install the application is described with the following commands:

```
$ git clone https://github.com/AlexeyAB/Yolo_mark.git
$ cmake .
$ make
$ ./linux_mark.sh
```

The last command opens the application for the user to start labeling. This application has a couple of dependencies that need to be installed. These are OpenCV (version 2.X or version 3.X) and CMAKE. The labeling process is very simple, and it only requires the user to use the cursor to enclose the target on the image which saves by pressing the next image button on the application. This process is shown in the following figures:



(a) Without bod (b) With box

Figure 5: Yolo Mark software

After finishing the labeling process each of the images should have the labels stored in a text file with the same name.

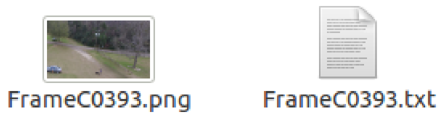


Figure 6: Image and label

To divide the dataset into training and validation the python script `divide_dataset.py` was used. This script can be downloaded from the Team 307 Github with the same commands used for the Pi setup. In this script, the path to the images has to be specified in the file and in the command:

```
$ python3 divide_dataset.py /path/to/images/
```

The next step would be training the neural network. To train the neural network the darknet repository is needed. It can be downloaded and installed using the following commands:

```
$ git clone https://github.com/AlexeyAB/Yolo_mark.git
$ make
```

After installing darknet pre-trained weights need to be downloaded for training. The yolov3 tiny weights can be downloaded from the following link:

- https://pjreddie.com/media/files/darknet53.conv.74

This file need to be placed in the darknet folder as it will be referenced in the training command. After downloading the pre-trained weights take the obj1.data, obj.names and the yolov3-tiny1.cfg from the Team 307 Github. The obj1.data file needs to be edited to add the new path for the two files created from the `divide_dataset.py` script. We recommend putting the obj1.data and obj.names file on the following path:

- /darknet/data/

The yolov3-tiny1.cfg should be put in the main darknet folder. After finishing with these files training can be started by running the following command:

```
$ ./darknet detector train data/obj1.data yolov3-tiny1.cfg darknet53.conv.74
```

This process will take a lot of time as it will be saving the trained weights on the backup folder in the darknet directory. To know when to stop training the loss value should be constantly monitored. The loss value can be seen in the following line which comes from the output of the training command:

- 9002: 0.211667, 0.060730 avg, 0.001000 rate, 3.868000 seconds, 576128 images Loaded: 0.000000 seconds

The 0.060730 avg represents the loss. An optimal value for this parameter should be between 5 and 0.05. When this value doesn't change for multiple training cycles the training should be stopped. After stopping training, the resulting weights can be tested for precision, recall, IoU and mAP values using the following command. The `<weight_number>` should be the exact number of the weights that's going to be tested.

```
$ ./darknet detector map data/obj1.data yolov3-tiny1.cfg backup\
> yolov3-tiny1_<weight _number>.weights
```

After testing several weights choose the one with the most optimal values and that will be the weight that will be converted to be used in the Raspberry Pi environment. The process to convert the weights to be used on the Pi with the NCS2 stick is done with the help of the OpenVINO-YoloV3 Github. To install OpenVINO the following commands need to be executed:

```
$ curl -sc /tmp/cookie "https://drive.google.com/uc?export=
>␣download&id=1ciX7cHqCh8lLFYI0HKkhC3r_fMirrlKk"/dev/null
$ curl -sc /tmp/cookie"<INSERT_LINK_HERE>"
```

Where the link is: `https://drive.google.com/uc?export=download&id=1NFt6g6Zkn eHioU2P7rUJ8BFpQhIazbym>/dev/null`

```
$ CODE = "$(awk␣'/_warning_/␣{print␣$NF}'␣/tmp/cookie)"
$ curl -Lb /tmp/cookie "<INSERT_LINK_HERE>" -o
> l_openvino_toolkit_raspbi_p_2019.1.133.tgz

$ tar -zxf l_openvino_toolkit_p_2019.1.133.tgz
$ rm l_openvino_toolkit_p_2019.1.133.tgz
$ cd l_openvino_toolkit_p_2019.1.133
$ sudo -E ./install_openvino_dependencies.sh
$ sudo ./install_GUI.sh
```

After installing OpenVINO the model optimizer needs to be configured. This application is used convert and optimize the trained weights. The commands that need to be executed in order to configure this application are shown below:

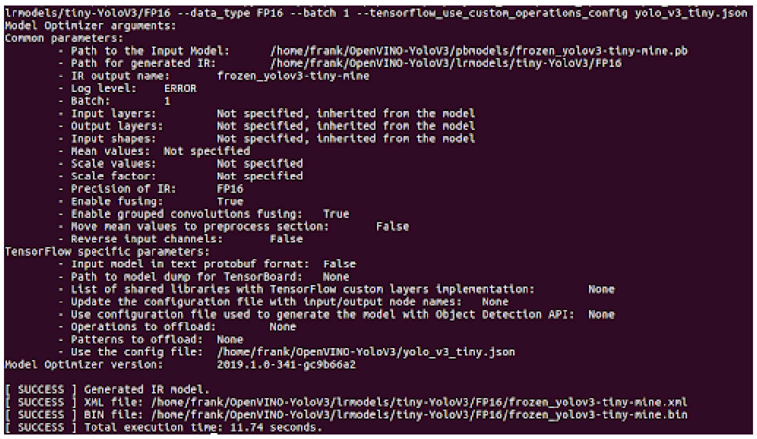
```
$ cd /opt/intel/openvino/install_dependencies/
$ sudo -E ./install_openvino_dependencies.sh
$ nano ~/.bashrc
$ source /opt/intel/openvino/bin/setupvars.sh
$ source ~/.bashrc
$ cd /opt/intel/openvino/deployment_tools/model_optimizer/install_prerequisites/
$ sudo ./install_prerequisites.sh

$ cd OpenVINO-YoloV3.git
$ cp ~/darknet/backup/ yolov3-tiny1_<weight _number>.weights weights
$ echo 'person' > obj.names
$ python3 convert_weights_pb.py \
$ --class_names obj.names \
$ --weights_file weights/ yolov3-tiny1_<weight _number>.weights \
$ --data_format NHWC \
$ --tiny \
$ --output_graph pbmodels/ yolov3-tiny1_<weight _number>.pb
```

The commands executed will create a file called `yolov3-tiny1_<weight _number>.pb`. This file is the .weights file that was converted to TensorFlow format. From the TensorFlow format we need to convert the .pb file the OpenVINO format. This is done by executing the following set of commands:

```
$ sudo python3 /opt/intel/openvino/deployment_tools/model_optimizer/mo_tf.py
$ --input_model pbmodels/yolov3-tiny1_<weight _number>.pb \
$ --output_dir lrmodels/tiny-YoloV3/FP16 \
$ --data_type FP16 \
$ --batch 1 \
$ --tensorflow_use_custom_operations_config yolo_v3_tiny.json
```

The terminal should output a series of commands similar to the ones on the figure below:



```
lrmodels/tiny-YoloV3/FP16 --data_type FP16 --batch 1 --tensorflow_use_custom_operations_config yolo_v3_tiny.json
Model Optimizer arguments:
Common parameters:
        - Path to the Input Model:      /home/frank/OpenVINO-YoloV3/pbmodels/frozen_yolov3-tiny-mine.pb
        - Path for generated IR:        /home/frank/OpenVINO-YoloV3/lrmodels/tiny-YoloV3/FP16
        - IR output name:       frozen_yolov3-tiny-mine
        - Log level:    ERROR
        - Batch:        1
        - Input layers:         Not specified, inherited from the model
        - Output layers:        Not specified, inherited from the model
        - Input shapes:         Not specified, inherited from the model
        - Mean values:  Not specified
        - Scale values:         Not specified
        - Scale factor:         Not specified
        - Precision of IR:      FP16
        - Enable fusing:        True
        - Enable grouped convolutions fusing:   True
        - Move mean values to preprocess section:       False
        - Reverse input channels:       False
TensorFlow specific parameters:
        - Input model in text protobuf format:  False
        - Path to model dump for TensorBoard:   None
        - List of shared libraries with TensorFlow custom layers implementation:        None
        - Update the configuration file with input/output node names:   None
        - Use configuration file used to generate the model with Object Detection API:  None
        - Operations to offload:        None
        - Patterns to offload:  None
        - Use the config file:  /home/frank/OpenVINO-YoloV3/yolo_v3_tiny.json
Model Optimizer version:        2019.1.0-341-gc9b66a2

[ SUCCESS ] Generated IR model.
[ SUCCESS ] XML file: /home/frank/OpenVINO-YoloV3/lrmodels/tiny-YoloV3/FP16/frozen_yolov3-tiny-mine.xml
[ SUCCESS ] BIN file: /home/frank/OpenVINO-YoloV3/lrmodels/tiny-YoloV3/FP16/frozen_yolov3-tiny-mine.bin
[ SUCCESS ] Total execution time: 11.74 seconds.
```

Figure 7: Final weights and conversion output

After this process is done the OpenVINO weights will be saved in the directory specified in the “path to for generated IR” described in the commands above. In that directory there will be four files, the weights are the files ending in .bin and .xml. This finishes the process of training the neural network for the use in the Raspberry Pi in combination with the NCS2 stick.

5 Operation

5.1 Operating the Drone

To operate the drone, connect the battery and the Pixhawk. The Pixhawk will power up upon battery connection. To connect a laptop device to the drone, the RDF900 modem must first be connected to the laptop and the QGroundControl program must be started.

(a) Verify GPS health by noting bright GREEN, flashing LED on Pixhawk. A bright BLUE flashing LED will flash if there is no GPS lock.

(b) Verify necessary flight plan settings at this point

(c) Verify data transmission by noting change in attitude, heading on the ground station laptop.

(d) Have qualified pilot power up the RC transmitter, select the correct vehicle, and verify connection by noting the green LED on the X8R receiver.

(e) Make sure all persons are clear of the prop from this point, until the drone is disarmed.

(f) To fully arm the vehicle, flip the SH switch to the up position on the Taranis transmitter. When the craft is fully armed, the bright GREEN LED(or bright BLUE for no GPS lock) will no longer be flashing.

(g) Test the motors to verify readiness by raising the throttle stick slightly and returning it to low.

(h) Complete the flight mission.

(i) After the vehicle is on the ground, flick the SH switch on the Taranis transmitter until the bright GREEN LED (or bright BLUE LED for no GPS lock) on the Pixhawk begins flashing. The vehicle is now disarmed and may be approached.

(j) Power off the vehicle by unplugging the battery.

5.2 Operating the Neural Network

To run the neural network, access to the drone Pi is needed. After accessing the Raspberry Pi open a terminal and input the following commands:

```
$ cd /home/pi/OpenVINO-YoloV3/
$ python3 tiny_test.py -d MYRIAD
```

These two commands should run the python script and create a window where the camera output can be seen. When there is a person in the camera feed the video output should show the person with bounded boxes around it and the percentage of confidence that the object identified is a person. This process is shown in the figure below:

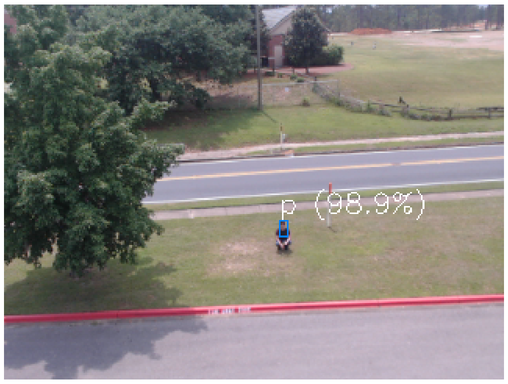


Figure 8: Output from the camera feed

5.3 Operating the Communication System

To start the communication system, the user must run one command on each of the Pi's. To start the transmitter Pi, the user should execute:

```
$ sudo python3 radio_Tx_v3.py
```

For the receiver, the user must execute the following command:

```
$ sudo python3 radio_Rx_v3.py
```

The Pi that's transmitting should output the number of packages that its sending, the image size, and the progress of the transmission. The receiver Pi should output on the terminal the number of packages received and the RSSI of each package.

6 Troubleshooting

- Drone does not operate within the specified maximum range
  - Check for obstructions between drone and ground station
  - Make sure the Yagi antenna is pointed towards the drone as much as possible
- Terminal is stuck saying "waiting for package"
  - Restart `radio_Tx_v3.py` and `radio_Rx_v3.py`
  - Make sure the Yagi antenna is oriented in the correct way
- Failing to get a GPS signal
  - Ensure the flight controller has GPS lock. The LED will flash bright BLUE if the flight controller does not have GPS lock
  - Once the flight controller has gained GPS lock, the LED will start flashing bright GREEN

7 References

[1] AlexeyAB. *AlexeyAB/Yolo_mark*. Apr. 2019. URL: https://github.com/AlexeyAB/Yolo_mark.

[2] *LoRa Raspberry Pi Setup | LoRa and LoRaWAN Radio for Raspberry Pi | Adafruit Learning System*. URL: https://learn.adafruit.com/lora-and-lorawan-radio-for-raspberry-pi/rfm9x-raspberry-pi-setup.

[3] PINTO0309. *PINTO0309/OpenVINO-YoloV3*. Apr. 2019. URL: https://github.com/PINTO0309/OpenVINO-YoloV3.

[4] Joseph Redmon. URL: https://pjreddie.com/darknet/.