CONCEPT:

The idea of the task is to make the drone detect a particular image placed on top of a moving robot and make it land accordingly on top of the image. So the task involves 3 major tasks:

- Drone and robot synchronization: To aid the drone with landing, the robot can either stop or move at a slower rate after getting a landing signal from the drone.
- Image Recognition: The drone much fly at a particular altitude/speed in order to detect an image since the lag in the drone camera is high. It should detect the designated symbol or image when it is required to land.
- Landing scheme: After detecting the image, it should mark a frame within which it can land and slowly make the landing.

IMAGE RECOGNITION:

To install Mission Planner for Ubuntu:

1) Install VirtualBox

- To install Virtual Box visit https://www.virtualbox.org/wiki/Downloads
- Choose the Windows platform package
- Follow the on-screen instructions to install the package

2) Download the latest version of Ubuntu Desktop from the link: https://ubuntu.com/download/desktop and the downloaded file is a **.iso** file

3) Open VirtualBox and follow the instructions to install Ubuntu. The **.iso** file generated during step 2 will be required during this installation. For Reference, visit this YouTube link: https://www.youtube.com/watch?v=QbmRXJJKsvs

4) Finally, to install the Mission Planner on Ubuntu: https://www.youtube.com/watch?v=KC7_X7IsVCI

To begin with how drone flight can be simulated and visualized on Mission Planner, the following link gives the right start: https://www.youtube.com/watch?v=TFDWs_DG2QY

The OpenCV library in Python was used for the image recognition.

Bash script to install OpenCV on Ubuntu:

1) Install OpenCV

Step 0: Select the OpenCV version to install

```
echo "OpenCV installation by learnOpenCV.com"
# Define OpenCV Version to install
cvVersion="master"
```

Clean build directories and create installation directory:

```
# Clean build directories
rm -rf opencv/build
```

```
rm -rf opencv_contrib/build
```

Storing currently working variable in cwd variable:

```
# Save current working directory
cwd=$(pwd)
```

Step 1: Update packages

```
sudo apt -y update
sudo apt -y upgrade
```

Step 2: Install OS Libraries

```
sudo apt -y remove x264 libx264-dev

## Install dependencies
sudo apt -y install build-essential checkinstall cmake pkg-config yasm
sudo apt -y install git gfortran
sudo apt -y install libjpeg8-dev libpng-dev


sudo apt -y install software-properties-common
sudo add-apt-repository "deb http://security.ubuntu.com/ubuntu xenial-
security main"
sudo apt -y update


sudo apt -y install libjasper1
sudo apt -y install libtiff-dev


sudo apt -y install libavcodec-dev libavformat-dev libswscale-dev
libdc1394-22-dev
sudo apt -y install libxine2-dev libv4l-dev
cd /usr/include/linux
sudo ln -s -f ../libv4l1-videodev.h videodev.h
cd "$cwd"


sudo apt -y install libgstreamer1.0-dev libgstreamer-plugins-base1.0-
dev
sudo apt -y install libgtk2.0-dev libtbb-dev qt5-default
sudo apt -y install libatlas-base-dev
sudo apt -y install libfaac-dev libmp3lame-dev libtheora-dev
sudo apt -y install libvorbis-dev libxvidcore-dev
sudo apt -y install libopencore-amrnb-dev libopencore-amrwb-dev
sudo apt -y install libavresample-dev
sudo apt -y install x264 v4l-utils


# Optional dependencies
sudo apt -y install libprotobuf-dev protobuf-compiler
sudo apt -y install libgoogle-glog-dev libgflags-dev
sudo apt -y install libgphoto2-dev libeigen3-dev libhdf5-dev doxygen
```

Step 3: Install Python Libraries

```
sudo apt -y install python3-dev python3-pip
sudo -H pip3 install -U pip numpy
sudo apt -y install python3-testresources
```

Install virtualenv and virtualenvwrapper modules to create Python virtual environments

```
cd $cwd
############ For Python 3 ############
# create virtual environment
python3 -m venv OpenCV-"$cvVersion"-py3
echo "# Virtual Environment Wrapper" >> ~/.bashrc
echo "alias workoncv-$cvVersion=\"source $cwd/OpenCV-$cvVersion-
py3/bin/activate\"" >> ~/.bashrc
source "$cwd"/OpenCV-"$cvVersion"-py3/bin/activate


# now install python libraries within this virtual environment
pip install wheel numpy scipy matplotlib scikit-image scikit-learn
ipython dlib


# quit virtual environment
deactivate
```

Step 4: Download opencv and opencv_contrib

```
git clone https://github.com/opencv/opencv.git

cd opencv
git checkout $cvVersion
cd ..


git clone https://github.com/opencv/opencv_contrib.git
cd opencv_contrib
git checkout $cvVersion
cd ..
```

Step 5: Compile and install OpenCV with Contrib modules

```
cd opencv
mkdir build
cd build


cmake -D CMAKE_BUILD_TYPE=RELEASE \
            -D CMAKE_INSTALL_PREFIX=$cwd/installation/OpenCV-
"$cvVersion" \
            -D INSTALL_C_EXAMPLES=ON \
            -D INSTALL_PYTHON_EXAMPLES=ON \
            -D WITH_TBB=ON \
```

```
            -D WITH_V4L=ON \
            -D OPENCV_PYTHON3_INSTALL_PATH=$cwd/OpenCV-$cvVersion-
py3/lib/python3.5/site-packages  \
        -D WITH_QT=ON \
        -D WITH_OPENGL=ON \
        -D OPENCV_EXTRA_MODULES_PATH=../../opencv_contrib/modules  \
        -D BUILD_EXAMPLES=ON ..


make -j4
make install
```

2) To use OpenCV in C++

Using CMakeLists.txt

```
cmake_minimum_required(VERSION 3.1)
# Enable C++11
set(CMAKE_CXX_STANDARD 11)
set(CMAKE_CXX_STANDARD_REQUIRED TRUE)
```

Set OpenCV_DIR

```
SET(OpenCV_DIR <OpenCV_Home_Dir>/installation/OpenCV-
master/lib/cmake/opencv4)
```

Replace OpenCV_Home_Dir with correct path

```
SET(OpenCV_DIR /home/hp/OpenCV_installation/installation/OpenCV-
master/lib/cmake/opencv4)
```

```
mkdir build && cd build
cmake ..
cmake --build . --config Release
```

3) To use OpenCV in Python

```
workon OpenCV-master-py3
```

```
ipython
import cv2
print(cv2.__version__)
```

- **Image Detection using Multi-Scale Template Matching:**

  - **Template Matching Concept:**

    Template matching is a technique for finding areas of an image that are similar to a patch (template). A patch is a small image with certain features. The goal of template matching

is to find the patch/template in an image. To find it, the user has to give two input images: Source Image (S) – The image to find the template in and Template Image (T) – The image that is to be found in the source image. It is basically a method for searching and finding the location of a template image in a larger image. The idea here is to find identical regions of an image that match a template we provide, giving a threshold. In cases where almost identical templates are to be searched, the threshold should be set high. (t>=0.8)

- **Template Matching using OpenCV:**

  Template Matching is a method for searching and finding the location of a template image in a larger image. OpenCV comes with a function **cv2.matchTemplate()** for this purpose. It simply slides the template image over the input image (as in 2D convolution) and compares the template and patch of input image under the template image. Several comparison methods are implemented in OpenCV. (You can check docs for more details). It returns a grayscale image, where each pixel denotes how much does the neighborhood of that pixel match with template. If input image is of size *(WxH)* and template image is of size *(wxh)*, output image will have a size of *(W-w+1, H-h+1)*. Once you got the result, you can use **cv2.minMaxLoc()** function to find where is the maximum/minimum value. Take it as the top-left corner of rectangle and take *(w,h)* as width and height of the rectangle. That rectangle is your region of template. The limitations are: 1. Pattern occurrences have to preserve the orientation of the reference pattern image(template). 2. As a result, it does not work for rotated or scaled versions of the template as a change in shape/size/shear etc. of object w.r.t. template will give a false match. 3.The method is inefficient when calculating the pattern correlation image for medium to large images as the process is time consuming. To avoid the issue caused by the different sizes of the template and original image we can use multi-scaling. In case where, just because the dimensions of your template do not match the dimensions of the region in the image you want to match, does not mean that you cannot apply template matching.

- **Multi- Scale Template Matching:**

  The process of Multi scaling is as follows:

  1. Loop over the input image at multiple scales (i.e. make the input image progressively smaller and smaller).
  2. Apply template matching using cv2.matchTemplate and keep track of the match with the largest correlation coefficient (along with the x, y-coordinates of the region with the largest correlation coefficient).
  3. After looping over all scales, take the region with the largest correlation coefficient and use that as your “matched” region.

Type the given command in the Python command prompt in order to view the output:

```
python match.py --template ub_logo.png --images images
```