IIT Dharwad

Sohan Anisetty
Akhil Manoj

Installation:

Installing ROS:

Setup your sources.list

```
sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu $(lsb_release -sc)
main" > /etc/apt/sources.list.d/ros-latest.list'
```

Set up your keys

```
sudo apt-key adv --keyserver 'hkp://keyserver.ubuntu.com:80' --recv-key
C1CF6E31E6BADE8868B172B4F42ED6FBAB17C654
```

Update packages and install ROS

```
sudo apt update
sudo apt install ros-melodic-desktop-full
```

Setup the environment

```
echo "source /opt/ros/melodic/setup.bash" >> ~/.bashrc
source ~/.bashrc
```

Dependencies

```
sudo apt install python-rosdep python-rosinstall
python-rosinstall-generator python-wstool build-essential
```

Rosdep

```
sudo apt install python-rosdep
sudo rosdep init
rosdep update
```

Installing Ardupilot:

Installing Ardupilot and MAVProxy

Clone ArduPilot

In home directory:

```
cd ~
sudo apt install git
git clone https://github.com/ArduPilot/ardupilot.git
cd ardupilot
git checkout Copter-3.6
git submodule update --init --recursive
```

Install dependencies:

```
sudo apt install python-matplotlib python-serial python-wxgtk3.0
python-wxtools python-lxml python-scipy python-opencv ccache gawk
python-pip python-pexpect
```

Use pip (Python package installer) to install mavproxy:

```
sudo pip install future pymavlink MAVProxy
```

Open ~/.bashrc for editing:

```
gedit ~/.bashrc
```

Add these lines to end of ~/.bashrc (the file open in the text editor):

```
export PATH=$PATH:$HOME/ardupilot/Tools/autotest
export PATH=/usr/lib/ccache:$PATH
```

Save and close the text editor. Reload ~/.bashrc:

```
. ~/.bashrc
```

Gazebo and Plugins

Gazebo

Setup your computer to accept software from http://packages.osrfoundation.org:

```
sudo sh -c 'echo "deb
http://packages.osrfoundation.org/gazebo/ubuntu-stable `lsb_release -cs`
main" > /etc/apt/sources.list.d/gazebo-stable.list'
```

Setup keys:

```
wget http://packages.osrfoundation.org/gazebo.key -O - | sudo apt-key add -
```

Reload software list:

```
sudo apt update
```

Install Gazebo:

```
sudo apt install gazebo9 libgazebo9-dev
```

Install Gazebo plugin for APM (ArduPilot Master) :

```
cd ~
git clone https://github.com/khancyr/ardupilot_gazebo.git
cd ardupilot_gazebo
```

```
git checkout dev
```

build and install plugin

```
mkdir build
cd build
cmake ..
make -j4
sudo make install
```

```
echo 'source /usr/share/gazebo/setup.sh' >> ~/.bashrc
```

Set paths for models:

```
echo 'export GAZEBO_MODEL_PATH=~/ardupilot_gazebo/models' >> ~/.bashrc
. ~/.bashrc
```

Installing MAVROS

Instructions for installing MAVROS can be found here but in short involve running the following command

```
sudo apt-get install ros-melodic-mavros ros-melodic-mavros-extras
wget
https://raw.githubusercontent.com/mavlink/mavros/master/mavros/scripts/inst
all_geographiclib_datasets.sh
chmod a+x install_geographiclib_datasets.sh
./install_geographiclib_datasets.sh
```

For ease of use on a desktop computer, please also install RQT

```
sudo apt-get install ros-melodic-rqt ros-melodic-rqt-common-plugins
ros-melodic-rqt-robot-plugins
```

We recommend using caktin_tools instead of the default catkin_make as it is more powerful

```
sudo apt-get install python-catkin-tools
```

Adding export lines to bashrc : Add these lines to the end of bashrc file.

```
export
GAZEBO_MODEL_PATH=/catkin_ws/src/interiit21/models:/ardupilot_gazebo/models

export
GAZEBO_PLUGIN_PATH=$GAZEBO_PLUGIN_PATH~/opt/ros/melodic/lib:~/usr/lib/x86_6
4-linux-gnu

export
GAZEBO_RESOURCE_PATH=$GAZEBO_RESOURCE_PATH~/catkin_ws/src/interiit21/worlds
```

Note:

Remove the **gimbal_small_2d model** from **~/.gazebo/models** for the camera topics to be visible

Encountered Errors:

If you encounter this error gazebo exit code 255 run -

```
killall gzserver
```

Launching the simulation

Creating a workspace

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/
catkin_make
```

We have created a workspace with the name catkin_ws, Now we will install the obstacle

avoidance package which we have made. First go inside your src directory,

```
cd src
```

Then copy the package and execute catkin_make. That's it!
Your final system should have the final file structure :

```
Home
    -- ardupilot
    -- ardupilot_gazebo
    -- catkin_ws
        -- src
            -- interiit21
                -- launch
                -- worlds
                -- models
                -- scripts
```

The launch folder contains the necessary launch file to start the gazebo simulation

The models folder contains the iris quadcopter models in sdf and urdf format

The worlds folder contains the world configuration files

The scripts folder contains the obstacle avoidance scripts

launching the simulation

Open 2 terminals
In one launch ardupilot SITL using :

```
sim_vehicle.py -v ArduCopter -f gazebo-iris  --console
```

In the other launch the interiit_world1.launch file in the launch directory. Navigate to the launch directory and execute :

```
roslaunch interiit_world1.launch
```

Note!
Launch the avoidance script after you get 3d fix from the ardupilot SITL

To launch the script navigate to the scripts folder and execute :

```
rosrun interiit21 moveDrone.py
```

The drone should take off and start navigating the obstacle course

Libraries used

1. Numpy
2. Opencv
3. Cv_bridge

Algorithm used :

We have developed a custom ros server and services for movement. You can control the drones position (x,y,z,yaw) , velocities(x,y,z,yaw), attitude(roll, pitch ,yaw), arm, change modes, takeoff and land. This script is called simple.py and is in the scripts folder .

We initially attempted to use moveit motion planner for trajectory estimation and path planning. To do this we developed custom packages to interface between moveit, ROS, gazebo and mavros. Also since moveit was originally built for robotic arms we had to develop our own action server and custom libraries to convert moveit trajectories into mavros commands. The library can be found here. We consider the drone as a single body with a floating joint with the world frame to do planning. However due to large computational power required for this we abandoned this approach.

After this we moved onto an OpenCv approach. Here we take the depth image given by the depth sensor and binarise it such that obstacles are 1 and free path is 0. Then we do simple contour detection, take the largest area (and hence the path which is relatively free of obstacles). Then we take the centroid of the largest contour and align the drones heading such that its center of vision coincided with the centroid. Similar conditions exist for up and down movement of the drone. For landing, the algorithm does an aruco-marker detection check on every loop. This check returns True, along with the coordinates of the center of the marker, only when an aruco marker with ID 0 is detected. Now, the drone adjusts its position such that the coordinates of the center of the marker nearly coincides with the center of the image taken by the down-facing camera. When this error is small enough, the drone is commanded to land.