A REPORT

ON

SURVEILLANCE OF COVID-19 PATIENTS USING DRONES IN CITY:A SURVEY

BY

V. Keerthivasan	2018A8PS0480P
Rishit Patel	2018A7PS0189G
Karan Shetty	2018A7PS0111G

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE, PILANI

A PRACTICE SCHOOL – 1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI-JUNE 2020

A REPORT

ON

SURVEILLANCE OF COVID-19 PATIENTS USING DRONES IN CITY:A SURVEY

BY

V. Keerthivasan	2018A8PS0480P	B.E.(Hons). Electronics and Instrumentation Engineering
Rishit Patel	2018A7PS0189G	B.E.(Hons) Computer Science Engineering
Karan Shetty	2018A7PS0111G	B.E.(Hons) Computer Science Engineering

Prepared in partial fulfilment of the

Practice School-I Course Nos.

BITS C221/BITS C231/BITS C241

AT

CENTRAL ELECTRONICS ENGINEERING RESEARCH INSTITUTE, PILANI

A PRACTICE SCHOOL – 1 STATION OF

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

JUNE 2020

Acknowledgement

We would like to acknowledge and convey our appreciation to all the people who have supported us and guided us during our time at one of India's most esteemed research institutes, CEERI, Pilani.

Firstly, we would like to extend our thanks to Dr D.K. Aswal, Director of CSIR-CEERI, Pilani. For having facilitated the collaboration between BITS Pilani University and CEERI. We would also like to thank Mr Vinod Verma, coordinator of the PS program at CEERI, as, without his efforts, this would not have been possible.

We thank Dr Samarth Singh of CEERI, Pilani for giving us the opportunity to work on this project and his valuable mentorship.

Lastly, we would like to thank our PS Instructors, Mr Pawan Sharma and Dr Rakesh Warier for guiding us through the project.

BIRLA INSTITUTE OF TECHNOLOGY AND SCIENCE PILANI

(RAJASTHAN)

Practice School Division

Station: CEERI, Pilani. **Centre:** Pilani.

Duration: 18th May- 28th June, 6 weeks **Date of Start:** 18th May, 2020

Date of Submission: 26th June, 2020

Title of the Project: Surveillance of COVID-19 Patients using drones in city: A Survey

ID No./Name(s)/ Discipline(s)/of the student(s):

V. Keerthivasan	2018A8PS0480P	B.E.(Hons). Electronics and Instrumentation Engineering
Rishit Patel	2018A7PS0189G	B.E.(Hons) Computer Science
Karan Shetty	2018A7PS0111G	B.E.(Hons) Computer Science

Name(s) and designation(s) of the expert(s):

Dr. Samarth Singh - Scientist

Name(s) of the PS Faculty: Mr. Pawan Sharma, Dr Rakesh Warier.

Key Words: Deep Q-Learning, ArUco marker, OpenCV, Reward system, Camera plugins, Vision-based landing

Project Areas: Drone Navigation, Reinforcement Learning, Vision-based landing

Abstract: We aim to use autonomous drones for delivery, and reduce human interaction, so as to reduce the risk of infection. This will help in automizing the control and maintaining social distancing. We used open source packages for the autonomous drones and the path planning was done using Q-Learning, which maximized the rewards and took the appropriate action. The main goal of the project was to implement vision-based landing on an autonomous drone.

Signature(s) of Student(s) **Signature of PS Faculty**

Date **Date**

Table of Contents

Introduction

Drones, extensively used today in surveillance and remote sensing tasks, start to also span in delivery tasks. For such outdoor tasks, the global navigation satellite system (GNSS) is the major solution for navigation. This solution has proven to be efficient and accurate, but fails in denied GNSS environments. Moreover, when obstacles in the path are unknown, there are too many of them, or they are at not fixed positions, then building a safe flight plan becomes very challenging. The same applies in environments with a weak satellite signal, such as indoors, under tree canopy or under the bridge, where the current GNSS drone navigation may fail.

Artificial intelligence, a research area with increasing activity, can be used to overcome such challenges. Initially focused on robots and now mostly applied to ground vehicles, artificial intelligence can also be used to train drones. Reinforcement learning (RL) is the branch of artificial intelligence able to train machines. Reinforcement learning is inspired by a human's way of learning, based on trial and error experiences. In RL, agents are the computerized systems that learn, and the trial and error experiences are obtained by interacting with the environment. Using the information about the environment, the agent makes decisions and makes actions in discrete intervals known as steps. Actions produce changes in the environment and also a reward. A reward is a scalar value informing about the benefit or inconvenience of such action. The objective of the agent is to maximize the final reward by learning which action is the best for each state. The application of RL will provide drones with more intelligence, eventually converting them in fully-autonomous machines.

For an unmanned drone to successfully function, autonomous landing is a crucial capability. The structured nature of landing makes it suitable for vision-based state estimation and control. This vision-based landing can be facilitated by integrating a deep-Q algorithm with ROS packages, in order to receive feedback from the sensors and give an output action to the drone. This action will help the drone navigate from its start position to the landing point, without encountering any obstacles.

ROS-Important Concepts and Definitions

What is ROS ?

ROS(Robotic Operating System) is a huge software framework for development of Robotics Software. ROS is not an operating system.ROS can be said as a combination of plumbing (behind nodes and message passing), a rich and mature set of tools, a wide-ranging set of robot-agnostic capabilities provided by packages, and a greater ecosystem of additions to ROS.

1. **Nodes**:

A node is a process that performs computation. Can be executed in both python and C++. Nodes are important constituents in forming a package.

Nodes are not specific to the PubSub model but also Client-server model. 'Rosnode' is a command-line tool for displaying information about Nodes.

Nodes are combined together into a graph and communicate with one another using streaming topics and services.

2. **Messages**:

A message is a simple data structure, comprising typed fields.

They are communicated through topics and services. They are either simple data types like float, int or they might include structures and arrays. 'rosmsg' is a command-line tool for displaying information about messages.

3. **Topics**:

Topics are named buses over which nodes exchange messages. Topics are intended for unidirectional, streaming communication.'rostopic' is the command-line tool for interacting with ROS topics. It supports both TCP/IP and UD based message transports.

Content of the messages sent could be sensor data, motor control commands, state information, actuator commands, or anything else.'rostopic' is the command line tool for viewing topics in ROS.

4. Services:

A service represents an action that a node can take which will have a single result.Nodes advertise services and call services from one another. Services are often used for actions which have a defined beginning and end.'rosservice' and 'rossrv' are the command line tools for viewing any information regarding Service in ROS.

5. Publisher/Subscriber:

A publisher is a node that keeps publishing a message into a topic whereas a subscriber is a node that reads information from a topic.

6. Packages:

ROS uses packages to organize its programs. It has all the files that a specific ROS program contains; all its cpp files, python files, configuration files, compilation files, launch files, and parameters files. All those files in the package are organized with the following structure:

- **launch folder**: Contains launch files
- **src folder**: Source files (cpp, python)
- **CMakeLists.txt**: List of cmake rules for compilation
- **package.xml**: Package information and dependencies

7. Launch Files:

ROS uses launch files in order to execute programs.
All launch files are contained within a <launch> tag. Inside that tag, you can see a <node> tag, where we specify the following parameters:

1. **pkg="package_name"**: Name of the package that contains the code of the ROS program to execute
2. **type="python_file_name.py"**: Name of the program file that we want to execute
3. **name="node_name"**: Name of the ROS node that will launch our Python file
4. **output="type_of_output"**: Through which channel you will print the output of the Python file

8. Roscore:

Every time a new node is created the roscore service program provides it with all the information needed in order to form peer-to-peer connection with the other nodes. Since the roscore has such an important role his presence is mandatory in every ROS system and that is why the roscore is always the first program that has to run.

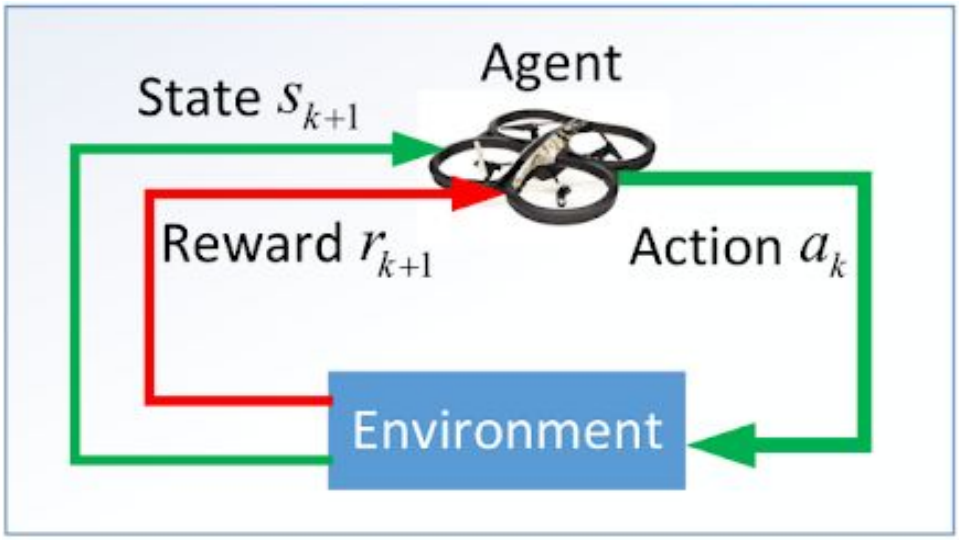
Q-Learning

Q-learning is a model-free reinforcement learning algorithm to learn a policy telling an agent what action to take under what circumstances. It does not require a model of the environment, and it can handle problems with stochastic transitions and rewards, without requiring adaptations.

Markov-Decision Process

Markov decision processes give us a way to formalize sequential decision making. This formalization is the basis for structuring problems that are solved with reinforcement learning.

In an MDP, we have a decision maker, called an agent, that interacts with the environment it's placed in. These interactions occur sequentially over time. At each time step, the agent will get some representation of the environment's state. Given this representation, the agent selects an action to take. The environment is then transitioned into a new state, and the agent is given a reward as a consequence of the previous action.



- At time t, the environment is in state St
- The agent observes the current state and selects action At
- The environment transitions to state St+1 and grants the agent reward Rt+1.
- This process then starts over for the next time step, t+1.It is important to Note that t+1 is no longer in the future, but is now the present. When we cross the dotted line on the bottom left, the diagram shows t+1 transforming into the current time step t so that St+1 and Rt+1 are now St and Rt.

Transition probability in MDP

If sets S and R are finite, the random variables Rt and St have well defined probability distributions. In other words, all the possible values that can be assigned to Rt and St have some associated probability. These distributions depend on the preceding state and action that occurred in the previous time step t−1.For example, suppose s′∈S and r∈R. Then there is some probability that St=s‘ and Rt=r. This probability is determined by the particular values of the preceding state s∈S and action a∈A(s). Note that A(s) is the set of actions that can be taken from state s. For all s′∈S, s∈S, r∈R, and a∈A(s), we define the probability of the transition to state s′ with reward r from taking action a in state s as:

P(s′,r|s,a)=Pr{St=s′,Rt=r|St−1=s,At−1=a}.

State-value function

The state-value function for policy π, denoted as vπ, tells us how good any given state is for an agent following policy π. In other words, it gives us the value of a state under π.

Formally, the value of state s under policy π is the expected return from starting from state s at time t and following policy π thereafter. Mathematically we define

$$V\pi(s) = E\pi[Gt \mid St = s] = E\pi[\infty\sum k = 0\gamma kRt + k + 1 \mid St = s]$$

The optimal policy has an associated optimal state-value function.We denote the optimal state-value function as v∗ and define as

v∗(s)=max Vπ(s) for all s∈S.

In other words, v∗ gives the largest expected return achievable by any policy π for each state.

Action-value function

Similarly, the action-value function for policy π, denoted as qπ, tells us how good it is for the agent to take any given action from a given state while following policy π. In other words, it gives us the value of an action under π

Formally, the value of action a in state s under policy π is the expected return from starting from state s at time t, taking action a, and following policy π thereafter. Mathematically, we define qπ(s,a) as

qπ(s,a)=Eπ[Gt|St=s,At=a]=Eπ[∞∑k=0γkRt+k+1|St=s,At=a].

Conventionally, the action-value function qπ is referred to as the Q-function, and the output from the function for any given state-action pair is called a Q-value. The letter “ Q” is used to represent the quality of taking a given action in a given state.The optimal policy has an optimal action-value function, or optimal Q-function, which we denote as q∗ and define as

$q_*(s,a)=\max q_\pi(s,a)$ for all $s \in S$ and $a \in A(s)$.

In other words, q_* gives the largest expected return achievable by any policy π for each possible state-action pair.

Bellman -optimality Equation

One fundamental property of q_* is that it must satisfy the following equation.

$q_*(s,a)=E[R_{t+1}+\gamma \max_{a'} q_*(s',a')]$

This is called the Bellman optimality equation. It states that, for any state-action pair (s,a)at time t, the expected return from starting in state s, selecting action a and following the optimal policy thereafter (AKA the Q-value of this pair) is going to be the expected reward we get from taking action a in state s, which is Rt+1, plus the maximum expected discounted return that can be achieved from any possible next state-action pair (s′,a′).

Since the agent is following an optimal policy, the following state s′will be the state from which the best possible next action a′ can be taken at time t+1.

Algorithm

The weight for a step from a state Δ t steps into the future is calculated as $\gamma \Delta t$ (where y is the discount factor) is a number between 0 and 1 and has the effect of valuing rewards received earlier higher than those received later (reflecting the value of a "good start"). γ may also be interpreted as the probability to succeed (or survive) at every step Δ t .

The algorithm, therefore, has a function that calculates the quality of a state-action combination:

$Q : S \times A \rightarrow R$

Before learning begins, Q is initialized to a possibly arbitrary fixed value (chosen by the programmer). Then, at each time t the agent selects an action a(t), observes a reward r(t),enters a new state s(t + 1) (that may depend on both the previous state s and the selected action), and Q is updated.

The core of the algorithm is a Bellman equation as a simple value iteration update, using the weighted average of the old value and the new information:

$$Q^{new}(s_t, a_t) \leftarrow \underbrace{Q(s_t, a_t)}_{\text{old value}} + \underbrace{\alpha}_{\text{learning rate}} \cdot \overbrace{\left(\underbrace{r_t}_{\text{reward}} + \underbrace{\gamma}_{\text{discount factor}} \cdot \underbrace{\max_a Q(s_{t+1}, a)}_{\text{estimate of optimal future value}} \right)}^{\text{new value (temporal difference target)}} \overbrace{- \underbrace{Q(s_t, a_t)}_{\text{old value}}}^{\text{temporal difference}}$$

Where r(t) is the reward received when moving from the state s(t) to the state s(t + 1) , and α is the learning rate ($0 < \alpha \leq 1$).

An episode of the algorithm ends when state s(t + 1)is a final or terminal state. However, Q-learning can also learn in non-episodic tasks If the discount factor is lower than 1, the action values are finite even if the problem can contain infinite loops.For all final states s' , Q (s ', a) is never updated, but is set to the reward value r observed for state s' . In most cases, Q (s' , a) can be taken to equal zero.

Softwares to be used:

- **Gazebo:**

The robot simulation environment which will be used in our project. It allows for the custom creation of robots and vehicles using pre-made models and also simulation of realistic environments using a physics engine like gravity, wind and other real-life conditions.

One feature of Gazebo that is important to this project is a robot and simulation plugin. A plugin is code used to customize the simulation environment, a vehicle sensor, or the vehicle itself. These plugins connect to the custom API built for Gazebo.

- **PX4 (Firmware):**

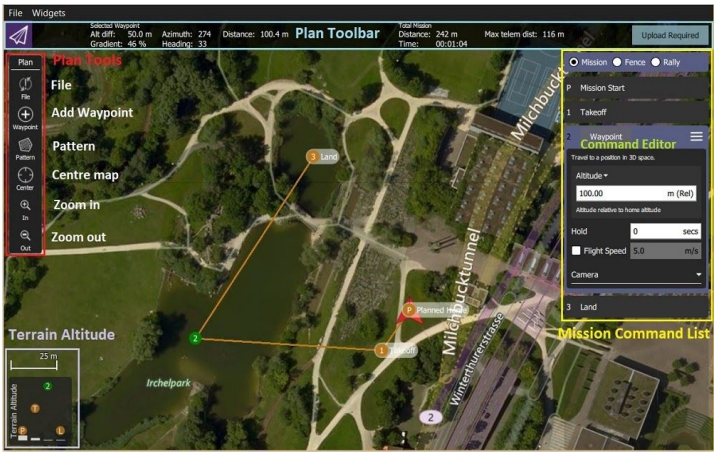
PX4 is an autopilot system that can fly or drive Unmanned Aerial or Ground Vehicles. It is loaded on simulated vehicle control hardware in our project. The autopilot must be paired with its counterpart ground control station, QGroundControl, in order to make a fully autonomous autopilot system.

PX4 communicates with the ground control station using the MAVLink communication protocol.

- **QGroundControl:**

A ground control system for use with the PX4 autopilot system. It provides full flight control and vehicle setup for vehicles with PX4. It displays the location of the vehicle, flight track, waypoints and vehicle instruments.

Another important quality of QGroundControl for this study is its ability to manage multiple vehicles. It has support for other autopilot systems as well as long as they also use MAVLink protocol for communication.



- **MAVLink:**

MAVLink (Micro Aerial Vehicle Link) is a protocol for communication that uses as messages a stream of bytes that has been encoded and is sent to the autopilot via USB serial or telemetry. The encoding procedure puts the packet into a data structure and sends it via the selected channel in bytes, adding some error correction alongside. MAVLink follows a modern hybrid publish-subscribe and point-to-point design pattern:

Data streams are sent / published as topics while configuration sub-protocols such as the mission protocol or parameter protocol are point-to-point with retransmission. It is used for communicating with drones and ground control systems (QGroundSystem). It can be used to transmit the orientation of the vehicle, its GPS location and speed.

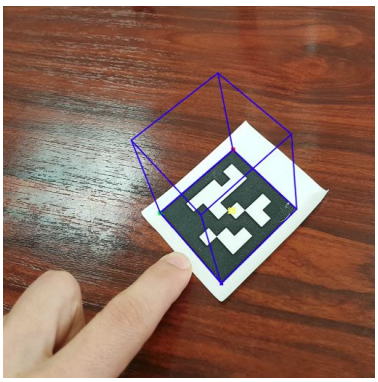
- **MAVROS:**

It is an extendable ROS package that provides a communication driver for autopilots and provides the MAVLink bridge for QGroundControl. MAVROS consists of nodes that each have a specific job of publishing certain types of information to the autopilot system and the ground control system.

This tool is important in transferring data back and forth from PX4 to QGroundControl to communicate the next action the drone must take in the simulation, in order to maximize its rewards. In particular, it allows the user to program and customize the parameters of the autopilot utilizing the ROS environment and replacing by all means any kind of GCS software.

- **ArUco:**

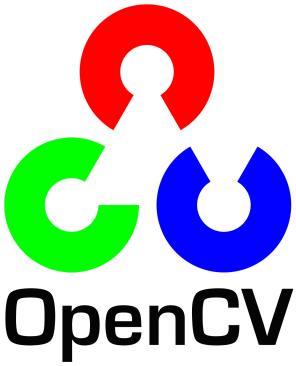
ArUco is an OpenSource library for detecting squared fiducial markers in images. Additionally, if the camera is calibrated, you can estimate the pose of the camera with respect to the markers.



The markers we will use for landing are called Fiducial markers. They are encoded objects that are used by computer vision systems as reference points in a scene. An aruco marker, a *fiducial marker,* is a binary square with black background and boundaries and a white generated pattern within it that uniquely identifies it(each marker has a unique ID).

These targets provide orientation, location and distance information when combined with accurate size information of the markers and calibrated camera information.The distance to target is obtained automatically through pose estimation of the markers.

- **OpenCV:**

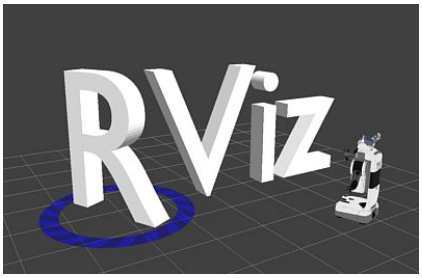


OpenCV(Open Source Computer Vision Library) is a library of programming functions mainly aimed at real-time computer vision.In order to perform any accurate Computer Vision work, you must first calibrate your camera.

OpenCV uses 'Camera Matrix' and 'Distortion Coefficients' matrices (collectively called intrinsics) for camera calibration.They are used to 'undistort' the raw image for accurate further processing, necessary to calculate the target angular offsets for precision landing and pose estimation used for accurate distance measurements.

- **RViz:**

RViz, abbreviation for ROS visualization, is a powerful 3D visualization tool for ROS. It allows the user to view the simulated robot model, log sensor information from the robot's sensors, and replay the logged sensor information.



RViz displays 3D sensor data from stereo cameras, lasers, Kinects, and other 3D devices in the form of point clouds or depth images. 2D sensor data from webcams, RGB cameras, and 2D laser rangefinders can be viewed in rviz as image data.

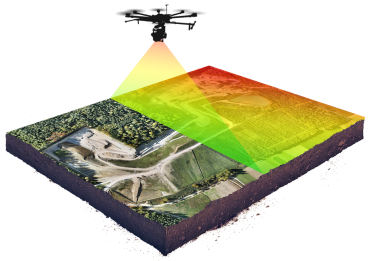
Sensors and Feedback:

- **LiDAR:**

LiDAR, in short for Light Detection and Ranging, can be considered as a laser distance sensor. It measures the range of targets through light waves from a laser instead of radio or sound waves.It has high measurement range and accuracy with ability to measure 3D structures.

It has a fast update rate which is suitable for fast-moving objects and uses small wavelengths as compared to sonar and radar.It is good at detecting small objects and applicable for usage in the day and night.

The cost is higher as compared to ultrasonic and IR for the LiDAR sensor.It is harmful to the naked eye as higher-end LiDAR devices may utilize stronger LiDAR pulses which may affect the human eye.



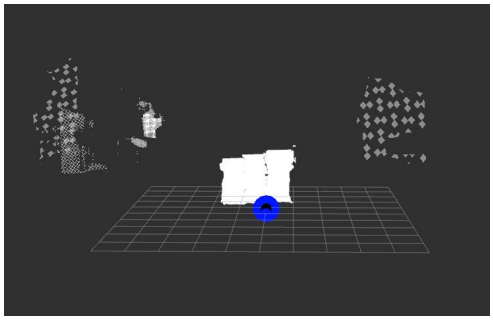
- **Stereocamera:**

A stereo camera is a camera that simultaneously photographs an object from a plurality of different directions using two cameras in the same manner as the principle that a person views an object. It is a camera that can measure information in the depth direction from the position information of the pixel of the camera.

It can detect objects of any distance and has high distance accuracy of near objects. Absolute distance from the camera can be acquired by 3D measurement .

A point cloud is a set of data points in space.Depth data is the information about depth taken from a sensor that the point cloud can express.

There are three main steps for pointcloud generation using stereocamera– disparity map generation,map disparity map to 3D points in camera frame and map 3D points in camera frame to drone body frame.

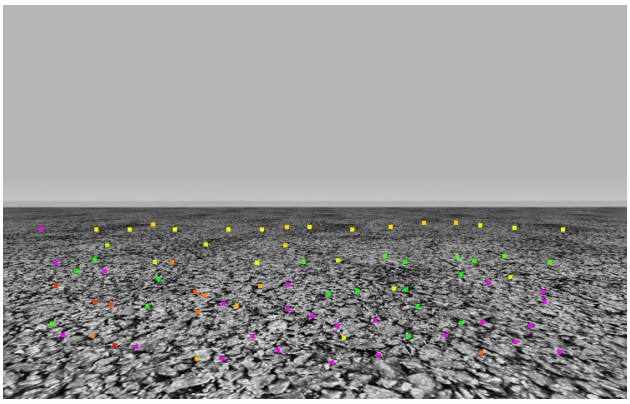


- **SLAM:**

SLAM is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.

SLAM is used to estimate the position of a vehicle as well as map its precise position. It is especially helpful in GPS denied environments like under the bridge, inside a room or crowded places

SLAM uses cameras as its primary sensor, which could be a monocular camera, stereo camera or RGBD camera.

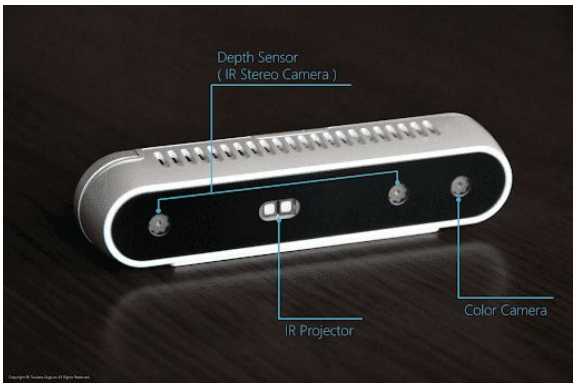


- **IntelRealSense:**

Intel RealSense technology basically consists of a processor for image processing, a module for forming depth images, a module for tracking movements and depth cameras. These cameras rely on deep scanning technology, which enables computers to see objects in the same way as humans do.

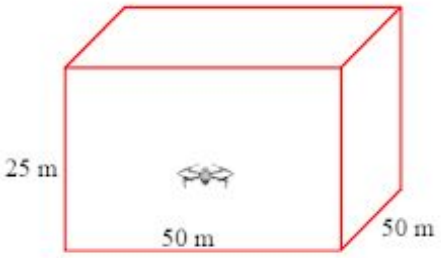
This camera is suitable for applications in robotics, in cases when no great accuracy and precision are required, but a general visual experience is more important

These cameras have three lenses: a conventional RGB camera, an infrared (IR) camera and an infrared laser projector. All three lenses jointly make it possible to assess the depth by detecting the infrared light that has been reflected from the object/body that is in front of it.



RL architecture:

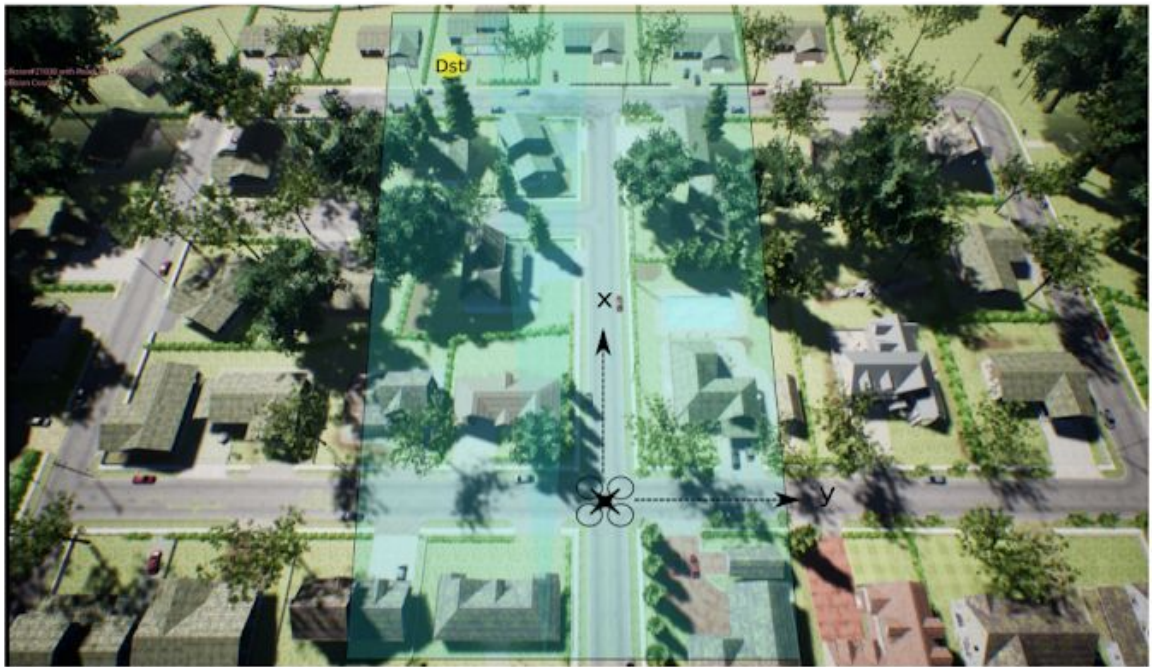
- The drone must fly inside the area delimited by the constraint space, that is, a virtual orthogonal box. We will be considering 4 walls- left, right, front and back.
- The limits of the Constraint Space: 25m height x 50m length x 50m width.
- We are the vertical descent to be constant, at 0.5m/s, thereby only focussing on the 2D part of the problem (XY plane)



State:

Consists of scalar values:

- The current location of the drone (px, py coordinates)
- The distance to the goal (dx, dy, dt, where dt is the Euclidean distance calculated from dx and dy)
- The distance to the Constraint Space boundaries in 2D (dxmin, dxmax, dymin, dymax)



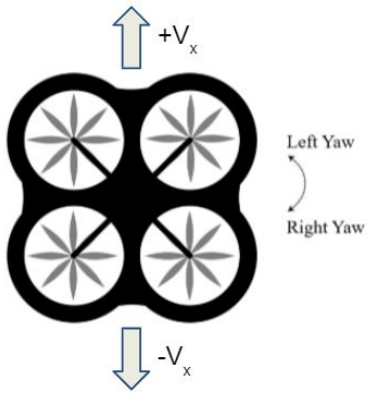
Reward:

+100	if terminal step (goal reached),
-100	if terminal step (failure: obstacle or constraint space),
-1 + Δdg	otherwise.

A terminal step returns a reward of +100 or −100 depending on whether the episode has respectively succeeded (goal reached) or failed.

Intermediate steps return a reward of −1 (to penalize delays) plus Δdg, that is, the distance-to-the-goal difference with respect to the previous step. This Δdg is used to stimulate actions that approach the goal.

Action:



Horizontal action set:

Description	**Value**
Turn left	30◦
Move forward	2 m/s
Turn right	30◦
Move backward	-2 m/s

The action will be done for 1 second, after which the decision will be taken.

Drone starts at 20m height and descent speed is 0.5m/s, irrespective of the action.

Goal is around 70-80m from the starting point.

Conclusion:

This project has made us understand the basics of ROS and the various open source packages and libraries that are present in ROS. It has basically helped us gain an insight as to how an autonomous drone works. We understood the integration of a deep-Q Algorithm with ROS, which takes inputs as distances from the various sensors and gives a control action as output to the drone. No human intervention is required once this has been established and the drone has learnt on its own,

We have created the idea of a drone which can reach an allotted target, on its own after trial and error in simulation. This is extremely useful to deliver supplies, especially now when people are in dire need of medicines and other essentials but can not go out to get them. This drone works using various open source packages along with some sensors attached to it, and effectively maps the shortest distance to the goal, without hitting any obstacles along the way.

In times like these, all of us have to keep maintaining social distance, and have to get back to our lives, keeping this global pandemic in mind and reducing as much human interaction as possible to reduce the chance of infection. Autonomous drones like these will help us in doing so.

References

- https://px4.io/
- https://www.ros.org/
- http://wiki.ros.org/mavros
- https://etd.ohiolink.edu/!etd.send_file?accession=wright1527192380795848&disposition=inline
- https://webthesis.biblio.polito.it/10933/1/tesi.pdf
- https://gaas.gitbook.io/
- https://docs.opencv.org/
- https://mavlink.io/en/
- https://deeplizard.com/
- https://www.researchgate.net/
- https://ieeexplore.ieee.org/
- https://github.com/goodrobots/vision_landing

Glossary

Deep Q-Learning:

The use of a neural network to approximate the Q-value function is called Deep Q-learning.The state is given as the input and the Q-value of all possible actions is generated as the output.

Disparity Map:

Disparity map refers to the apparent pixel difference or motion between a pair of stereo images. To experience this, try closing one of your eyes and then rapidly close it while opening the other. Objects that are close to you will appear to jump a significant distance while objects further away will move very little.

SLAM:

It stands for simultaneous localization and mapping.It is the computational problem of constructing or updating a map of an unknown environment while simultaneously keeping track of an agent's location within it.

Camera Matrix:

In computer vision a camera matrix or projection matrix is a matrix which describes the mapping of a pinhole camera from 3D points in the world to 2D points in an image

Distortion coefficients:

In computer vision, distortion coefficient is a 5x1 vector that contains the five parameters related to distortion of the lens in camera.

RGB camera:

The human eye is sensitive to red, green, and blue bands of light. Most standard drones come with cameras that capture the same RGB bands so the images they produce recreate almost exactly what our eyes see.