Designing autonomous drone for food delivery in Gazebo/Ros based environments

Hrishitva Patel
Binghamton university, hpatel51@binghamton.edu

Follow this and additional works at: https://scholar.smu.edu/engineering_compsci_research

Recommended Citation
https://scholar.smu.edu/engineering_compsci_research/6

This document is brought to you for free and open access by the Computer Science and Engineering at SMU Scholar. It has been accepted for inclusion in Computer Science and Engineering Research by an authorized administrator of SMU Scholar. For more information, please visit http://digitalrepository.smu.edu.
Designing autonomous drone for food delivery in Gazebo/Ros based environments

\*Note: This is an original work

1st Hrishitva Patel
Watson school of engineering(State University of New York, Binghamton)
Binghamton, New York
hpatel51@binghamton.edu

Abstract—There has been a growing global trend towards convenience, speed, and ease in delivery services, and this has been further accelerated by the COVID pandemic. With the ever-increasing demand for easily accessible deliveries and expanded delivery service coverage, it has become critical that innovations in this space be developed to further ensure the industry’s smooth operation. With the emergence of the COVID-19 pandemic, the inadequacies became more apparent, emphasizing the need to revolutionize and accelerate the trend in order to meet the increased demand. Drone delivery systems are of particular interest in this context because they can enable faster and more cost-effective delivery. This paper introduces a navigation system that simplifies the delivery of food parcels with independent drones. The system generates a path between the start and endpoints and controls the drone to fly this path based on its location obtained by planning the route through various sensors like LiDAR. The drone also avoids obstacles that come its way to achieve the intended goal, hence autonomously navigating its path. In the landing phase, marker information (ArUco Tag) uses a camera, and the drone software scale is integrated using an expanded Kalman filter algorithm to improve landing accuracy. The vector-based approach controls the drone to fly the desired path smoothly, minimizing vibrations or strong movements that could damage the transported package.

Index Terms—Motion planning, SLAM, ROS; cleaning, Autonomous Robot, navigation, food delivery, drone

I. INTRODUCTION

Autonomous robots have been studied for a long time, and the growing demand for automated solutions to real-world problems has increased research in the field. The online shopping marketplace has grown significantly in recent years, and since then, retail and mail groups have been attempting to make self-reliant drone transport a reality. Global online food shipping services reached 136.4 billion dollars in 2020, a 27% increase. Autonomous shipping presents itself as a very convenient alternative, particularly during periods of social isolation, such as the events that occurred in 2020 and 2021 as a result of the COVID-19 pandemic. [2] In situations like this, a large amount of bodily contact among people should be reduced with this viable option, particularly with humans from threat corporations. As a result, the transportation of goods with the help of self-sustaining quadrotors that avoid any bodily contact with the consumer adheres to World Health Organization (WHO) recommendations and hence prevents the virus from spreading. Amazon, UPS, and Alphabet, are moving closer to offering drone delivery services. According to Schneider (2020), Flight Forward, the American branch in charge of drone flights, has already obtained air carrier certification and is currently utilizing it in the supply of small programs with drones. Wing, an Alphabet division, launched the first small industrial delivery service in Christiansburg, Virginia. These accomplishments occurred in late 2019 and demonstrate that the international market for drone shipping is definitely gaining traction. In late 2020, Brazil’s National Aviation Administration (ANAC) granted the first authorization to a private company, Speed Bird, to conduct drone cargo checks in Brazilian cities. In this paper, I represent a technique for enabling self-sustaining drone deliveries. After producing a path among two points of interest, the drone’s region is obtained via its GPS (Global Positioning System), IMU (Inertial Measurement Unit), and a barometer. However, in drone transport, reproducible, secure landings in city regions are an essential task. In that respect, we suggest an Extended Kalman Filter (EKF) set of rules that fuses planar visual marker and ultra-wideband (UWB) localization strategies with the drone’s software program for posing estimation to enhance landing accuracy.[3] The visual localization uses the ArUco markers and the UWB localization is anticipated through multilateration with more than one UWB anchor over the landing place.

II. EASE OF USE

A. Problem Description

The problem addressed entails transporting small parcels among two points using a self-sufficient drone without receiving commands from a human pilot. Assuming that the landing stations are defined at safe places and the course is deliberate for above the top of homes and bushes, we have considered impediment-unfastened environments. Moreover, we assumed distances compatible with the drone’s endurance (maximum flight time). Besides, considering it’s far more essential for delivery drones to carry fragile objects without abrupt movements, we have followed a grasped transportation mechanism [4] that reduces the risks of vibrations or surprising bundle drops. In the beginning, residential regions may be the number one landing locality for transport obligations; therefore, the drone has to have the potential to land with
accuracy in confined and narrow regions to prevent unexpected injuries or accidents. The assessments have been completed with the landing platforms that would be available in mind.

III. Approach

To solve the problems of drone delivery, such as optimizing the work, reducing the delivery time, obstacle avoidance, and safe landing on a targeted point, I propose a solution in various parts: path planning/moving drone, localization and controlling the drone, and safest landings. This is accomplished using ROS and Gazebo simulations.

A. Hardware

After a successful result in the simulation, I implemented my code into the hardware to control the drone and deliver the parcel at the targeted point. All sensors that are used in the drone for online food delivery are GPS, IMU, and a barometer. These are liable for assisting in estimating the drone’s position in international coordinates (range and longitude), orientation, and peak. The drone’s flight board is responsible for transmitting statistics from the sensors to any other tool; likewise, it is responsible for receiving control commands and sending them to the brushless automobiles. In addition to the flight controller board that was required to be on the drone, a Jetson Nano2 is liable for statistics processing, course-making plans, and the quadrotor’s excessive-level control. For the drone landing, I used the ArUco tag/QR code that is horizontally placed to ensure precise accuracy for the drone’s location for the use of the approximate GPS. It’s far more necessary to use additional sensors to help the drone in making a safer and especially correct touchdown. For that, we verified the application of the following sensors in experiments for the use of the real drone: (i) a Raspberry Pi camera pointing downwards, together with ArUco markers, aggregate at the landing platform, and (ii) ultrawideband (UWB) gadgets anchored at the landing place and a tool of the equal kind attached to the drone. In both instances, it is feasible to achieve additional information on the drone’s function with the platform throughout the landing and improve localization by fusing these facts. Along with these, an Arduino Uno can also be embedded in the drone to monitor a change in wind pressure, weather conditions like rain, and sudden obstacles like birds and trees; so, in case of failed or delayed food orders, an emergency signal is sent, and the respective party can notify the end-user (customer) regarding the same.

B. Software

The operating system used is Ubuntu 18.04, together with ROS 1 and Gazebo/RVIZ (version: melodic). I use the parrot drone for this project. The parrot drone has four propellers, and for landing in the relevant place, I use the Aruco tag. The move_base package allows the drone to move forward on the path and reach the destination while avoiding obstacles. And for obstacle avoidance, I use the LiDAR sensor, which senses the object that comes in its path and estimates the drone’s position and orientation in an online fashion. For the identity and estimation of Aruco’s pose, specific algorithms from the Open CV are used. A set of rules makes use of the difference in the time of arrival of the sign for each tool to compute the drone’s role with respect to the landing site [4].

C. Methodology

Tools to be used:

- Ubuntu 18.04
- ROS Melodic version
- Python 2.7 or greater
- Gazebo 8 or 9
- RVIZ for visualization
- OPENCV
- LiDAR Sensor
- Radar
- Raspberry Pi
- Camera, Parrot Drone (or any four propeller drone)
- IMU
- Aruco Tag

- ROS is well known by means of its simulations in the discipline of robotics, and it is open-source. A robot running system comes with interfaces of message passing and equipment for simulation, i.e., Gazebo, Rviz, and has a number of packages that help simulate the challenge. Due to its several integration gears, it will now not make a lot of complexity. Exclusive libraries and programs are supplied by means of many humans to assist in sharing expertise. ROS is used to pass messages that give inter-technique communication, so it is generally referred to as middleware. Numerous centers are supplied by way of ROS, which helps researchers develop robot programs. In these studies, ROS is the main base of labor due to the publishing of messages on the topics, after which the subscribing of those messages from subjects takes place. This will be carried out with different parameters. ROS also provides inter-platform operability, modularity, and concurrent resource handling. ROS also helps us to create a virtual environment, generate robot models, implement the algorithms, and visualize it in the virtual world rather than implementing the whole system in the hardware itself. Therefore, the system can be improved accordingly, which provides us a better result when it is finally implemented in the hardware. Below are some important features that are provided by ROS:

  - Message passing interface: As the name suggests, it provides the capability of passing messages. The program exchanges the data with the help of linked systems of communication.

  - Package management: The ROS packages are designed with the help of ROS nodes. These packages consist of different files, like source codes, config, build files, etc. The build system helps to execute these packages. ROS provides this developed and systematic approach to management.

  - Low-level device control: There sometimes is the need to control low-level electronics to send data with
the help of serial ports and vice versa. This can only be attainable with the help of ROS.

- Distributed computing: When we are interfacing different sensors, the processing of data is very excessive from different robot sensors. In ROS, this high processing is distributed to clusters with many computing nodes. This helps in increasing the speed of processing data.

- Code reuse: The importance in terms of code reuse is very demanding, so the ROS growth factor increases day by day due to this sharing of data. The group of different packages is used to run one simulation. This is known as a meta-package, and both of these packages are shared or distributed.

- Scaling: ROS can be scaled for performing complex and difficult computations in robots.

ROS gives the combination of different message-passing interfaces, simulations, and visualizations of tools and capabilities. There are numerous capabilities of navigation, control, localization, manipulation, and mapping—all in all, a simulation environment that would more or less be similar to the real world. The robot is very much likely to behave similarly to the simulation when its application is tested in the real world. This results in the development of many robotics applications.

D. ROS file system

The organized way of files that includes different types of packages, messages, etc., is known by the ROS file system.

<table>
<thead>
<tr>
<th>Term</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Packages</td>
<td>It is an individual unit which contains codes, libraries, config files, executable files.</td>
</tr>
<tr>
<td>Messages</td>
<td>The description of message is stored in msg in package. To send the messages, these data structures are used. The extension is .msg.</td>
</tr>
<tr>
<td>Services</td>
<td>The srv folder in package describes the services having .srv extension. It is defined by request and the response data structure.</td>
</tr>
</tbody>
</table>

E. ROS Computational Graph

This computational graph in ROS is defined by the peer-to-peer network used to process the data. The features in the computational graph are given in the table:

<table>
<thead>
<tr>
<th>Feature</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Nodes</td>
<td>This is the executable part of the package which processes the data using functionalities.</td>
</tr>
<tr>
<td>ROS Master</td>
<td>A program which is known as ROS master is used to connect the ROS nodes to each other.</td>
</tr>
<tr>
<td>ROS nodes use ROS topics to communicate with each other. A ROS node can either publish on or subscribe to the ROS topic in the form of messages.</td>
<td></td>
</tr>
<tr>
<td>Services</td>
<td>We can send the service request to another node that provides the service with the help of message definition. This result is then sent as a reply. The node will wait for this result until it is received from a particular node.</td>
</tr>
<tr>
<td>Bags</td>
<td>In ROS bags data are stored so that it can be played back for ROS topics.</td>
</tr>
</tbody>
</table>

F. Gazebo

The Gazebo is a robot simulator. Gazebo enables a user to create complex environments and gives the opportunity to simulate the robot in the environment created. In Gazebo, the user can make the model of the robot and incorporate sensors in a three-dimensional space. In the case of the environment, the user can create a platform and assign obstacles to that. For the model of the robot, the user can use the URDF file and can give links to the robot. By giving the link, we can give the degree of movement for each part of the robot. The sample environment is shown below. The outdoor environment is created in this environment, having grass, trees, and the Aruco Tag, where objects were considered obstacles.

Gazebo is famous for its accuracy and efficiency in the simulation of complex robots in indoor and outdoor environments. High-quality graphics and renderings can be created with the help of Gazebo. The features of Gazebo are as follows:

- Dynamic Simulation
- Sensor Support
- Plugins
- Robot Models
- Command Line Tools

G. RVIZ

Rviz is used to visualize the URDF, sensors, and their data in 3D environments. It means to say if LiDAR is fitted on the robot in Gazebo, then we can visualize its scan data with the help of Rviz. With the help of scan data, a map can be built and
visualized and then used for autonomous navigation. This can represent IMU, laser scan, etc., and data graphically as well. The coordinates are called frames in Rviz. Many displays can be selected in Rviz to visualize the data from different sensors. An ADD button, when clicked, means we can display any data from sensors. The reference and ground position will be given with the help of a grid. From the sensor/Laser, scan data can be displayed. The position given by the program is displayed by Point Cloud. An axis can be displayed with the help of a reference point. The initial display of Rviz is given below:

The algorithm which I use in simulations is Hector Slam, move_base, navigation stack. First of all, I make the map of the environment using Hector Slam, in which I have the coordinates of my Aruco tag for my targeted point. Move_base helps the drone to fly and not fall on the ground while flying. The navigation stack helps to navigate the drone and find the shortest path to reach the goal. The navigation stack uses dwl, A*, and the EKF method, which is a combination of all these. For landing the drone, I use OpenCV, which helps to detect the Aruco tag and land it properly without falling.

H. LiDAR Sensor

LiDAR, also called Light Detection and Ranging Sensor, it is used in many robotics applications, due to its efficiency, reliability and robustness. The LiDAR sensor works on the principle which involves a laser pulse being sent from the transmitter and the reflected pulse being received by the receiver. Then the time is calculated between the transmission and receiving of the signal. This time is then used to calculate the distance of the objects around the LiDAR sensor. The LiDAR Sensor can perform 360 degree omni-direction laser range scanning for the surrounding environment. This data is used to perform mapping and also localization. Thus, it is a very important sensor in case of the autonomous navigation. Generally, it can have a range from 10-18 meters. Its publishing rate is 10 hz. Some of the LiDAR sensors also have compatibility with the ROS, which helps make the robot applications robust.

I. Simultaneous Localization and Mapping

In this section, we will take a look at simultaneous localization and mapping (SLAM). The basic purpose of SLAM is to build maps and localize the robot at the same time.[5] The robot can be run in two ways to make a map; either the robot moves autonomously or is controlled manually to move a robot for map-making. The main purpose is to make a map correctly. The basic way used to localize the robot is the use of odometry. In a robot, odometry consists of the combination of the displacement measured with the help of encoders for the estimation of orientation and position. This involves the odometry error in this way. SLAM works for the correct estimation of orientation and mapping. That is why this method is very difficult because errors of both the robot and map states affect each other. In order to navigate in any environment, we need a few key elements. For a robot, this is no different. What we need is:

- A map of the environment
- A route to take
- The current location on the map (localization)
- Obstacle avoidance. A map is mandatory for navigation

There are two ways to get a map:

- Use a pre-existing map
- Built by the robot itself (mapping process)

The mapping process is called SLAM (Simultaneous Localization and Mapping). Here, I am using a specific type of SLAM called Hector-mapping. Using the Laser Scanner, the drone can get information about the obstacles in the environment. The slam-hector-mapping node then processes this information. After that, it publishes the map topic. By using the map_server, map_saver, and slam_hectormapping node, the map is created of the world.

J. Autonomous Navigation

“Our outdated education system defines intelligence as a process of memorizing old answers to avoid errors. True intelligence is to learn to solve our problems to face greater challenges. True intelligence is about the joy of learning and not with the fear of failure.” — Robert Kiyosaki In this chapter, we are going to discuss autonomous navigation. In navigation tasks, the vehicle itself is able to plan its path and execute that plan without any human intervention. This path is fully based on the input coming from different sensors and the map of the environment, which is already created. This chapter relates to motion planning and deals with the partial view of the environment. The autonomous navigation problem is now solved using different algorithms [6]. In contrast with this, dynamic and unknown environments still represent a challenging task due to partial observable environments. Dynamic environments consist of different objects that can change their shape, position, and orientation, for example, doors, humans, animals, etc. There are three possibilities of assumptions that navigation systems consider when it plans movement [7].
1. Known Space Assumption- In this assumption, the whole area or region is supposed to be known to the robot via a map, which is already generated [8, 9]
2. Free Space Assumption- In this assumption, the unobserved space is supposed to be without any obstacles. It is often without a previous map. [10]
3. Unknown Space Assumption- In this assumption, the environment is totally observable. It is used for autonomous map-building tasks, not for navigation tasks. [9]

Most navigation systems have known space assumptions. In most cases, the environment is static or has even small dynamic obstacles, like humans existing, in cases of robot cleaning purposes, or the map doesn’t have any errors.

K. Algorithm

Various path planning algorithms are used in the robotics application for autonomous navigation task. Some of the path planning algorithms are RRT, PRT, A*, etc. The purpose of these planning algorithms is to find the best shortest path from the starting point A to finishing point B as quickly as possible. These algorithms use different ways to processes and navigate. In this project, A* algorithm is used. A* is an informed search algorithm, or a best-first search, meaning that it is formulated in terms of weighted graphs; starting from a specific starting node of a graph, it aims to find a path to the given goal node while incurring the smallest cost (least distance traveled, shortest time, etc.) [11]. It does this by maintaining a tree of paths, originating at the start node and extending those paths one edge at a time until its termination criterion is satisfied. The mentioned pseudocode describes the A* algorithm:

```
Input: Graph, start, goal
Output: path
Function A_star(start, goal):
    Set all nodes g(s) = ∞
    start = 0
    while open is not empty OR s is not goal do
        s = open.Pop()
        for all s' ∈ successors of s do
            if g(s) + cost(s, s') < g(s') then
                g(s') = g(s) + cost(s, s')
                open.insert(s', g(s') + h(s', goal))
        if PATH(start) = NONE
            break
        end
    end
    return PATH
End Function
```

L. ROS Navigation Stack

The ROS Navigation Stack is meant for maps, square or circular robots with a holonomic drive, and a planar laser scanner, all of which a drone robot has. It uses odometry, sensor data, and a goal pose to give safe velocity commands. The node move_base is where all the magic happens in the ROS Navigation Stack. It uses a global and local planner to accomplish the navigation goal. It manages communication within the navigation stack. Sensor information is gathered (sensor sources node), then put into perspective (sensor transformations node), then combined with an estimate of the position of the robot based on its starting position (odometry source node) [12]. This information is published so that move_base can calculate the trajectory and pass on velocity commands (through the base controller node). Right now, we are ready to go. Use the 2D pose estimate button at the top of your screen to localize the robot, then use the 2D Nav goal to give the drone a target position to move to. Next, watch the drone as it generates a path and tries to follow it.

M. Navigation Stack Architecture

The following Figure shows the architecture of the navigation stack of ROS.
Sensors provide the input values: sensor sources and odometry sources. Based on those two data sources, the algorithm calculates the sensor transformation, the pose estimate (AMCL), and maps the environment (map server). Thanks to all this data, the path planner can make decisions. Sensors are essential, and the LiDAR is the best sensor to get LaserScan or PointCloud data. Below is the explanation of the blocks used in the above architecture diagram of the navigation stack.

- **Odometry Source.** This contains the robot position and orientation data. Main sources include wheel encoders, IMU, and laser scanners, i.e., visual odometry. This odometry data is published on the navigation stack with an Odom node and contains a message type of nav_msgs/Odometry.
- **Sensor Source.** To map the environment, there is the need for laser scan data. These two data inputs combine to make the local and global cost maps [13]. The sensor used here is LiDAR. The data is in the type of sensor_msgs/LaserScan.
- **Sensor Transform/tf.** The robot-coordinated frames are published by the ROS tf node.
- **base controller.** This base controller converts the twist output messages into motor velocities. The other nodes include the AMCL and map server to allow the localization of the environment [13].

### N. Understanding move_base node

The package move_base has a node called the move_base node. The main objective of the node is to control the robot and move it from the start to the goal position with the help of navigation stack nodes. The move_base connects both the local and global plannners to decide the path trajectory to the goal. For obstacle avoidance, it connects the global costmap to the local costmap. The move_base node basically is an implementation of SimpleActionServer, which takes a goal pose with the message type (geometry_msgs/PoseStamped). We can send a goal position to this node using a SimpleActionClient node. The move_base node subscribes to the goal from a topic called move_base_simple/goal, which is the input of the Navigation stack, as shown in the previous diagram. When this node receives a goal pose, it links to components such as global_planner, local_planner, recovery_behavior, global_costmap, and local_costmap, which generates the output, which is the command velocity (geometry msgs/Twist) and sends it to the base controller for moving the robot for achieving the goal pose.

### O. ArUco

For the use of the marker detection technique for localization, ArUco markers were revealed and positioned on top of the touchdown platform. A camera attached to the drone, pointing downwards, gives pictures of the marker all through landing. ArUco markers have functions that facilitate their identification in the photograph, such as nicely-defined borders and high color contrast. In addition, the markers do not gift ambiguities in their orientation. Hence, specific OpenCV algorithms perceive the ArUco in addition to estimating the relative pose of the marker with respect to the digital camera. This final step is accomplished via fixing the hassle of PnP (attitude-n-point), which proposes to estimate the 3-dimensional pose of a calibrated camera that gives a fixed position of 3D points and their corresponding second projections on the camera aircraft.
ArUco markers can be detected from high altitudes. However, when the drone is approaching the platform, this ArUco is quickly lost by the camera. On the other hand, a smaller ArUco has the advantage of being detectable when the drone is close to the platform (if there is no high horizontal error), even though it is difficult to be detected at high altitudes. To improve the marker detection range at high and low altitudes, I considered a modified ArUco marker that has a smaller marker inside a larger one. The inclusion of the smaller marker may harm the detection of the larger one. Nonetheless, in our tests, this problem did not occur.

**Conclusion**

The use of delivery services is becoming more common around the world, a trend that has been accelerated by the COVID pandemic. Drone delivery systems are of particular relevance in this context since they provide faster, cheaper, pollution-free, and contactless delivery.

In this paper, I have given the proposed solution to this problem. I present a navigation method that includes localization, path planning, landing, and self-delivery. I created an external environment in Gazebo for moving the drone from one location to another. After creating the environment, I set up my drone in it and placed an Aruco tag. For drone navigation, I used a navigation stack. Using this package, I created an A* algorithm that aids the drone in locating the quickest path to the desired location. The desired location coordinates and surrounding landmark information can be fed to the Raspberry pi by retrieving them from an API call, made available by third party companies providing global maps like Google. A few sensors help in navigation, the most important of which is the LiDAR sensor, which helps detect and avoid objects in its path. I also attached the camera sensor below the drone for detecting the Aruco Tag. For detecting Aruco, I used an OPENCV algorithm, which helps detect the tag and helps in precise landing. A frame can be overlaid on the drone stimulation depicted on Gazebo to give it a realistic feel.

**Related Work**

Drone delivery is a new field that is gaining traction in academia and industry due to the numerous challenges that must be overcome in order to carry out successful missions in urban environments, such as trajectory planning, localization, steerage and manipulation, impediment avoidance, and a safe landing, especially when global localization is unavailable or unreliable. The exploration of drone-assisted parcel delivery is also motivated by the environmental advantages of aerial platforms over traditional truck delivery [4]. The point on the landing procedure illustrates a managed approach for a self-sufficient touchdown in a fixed platform that is totally based on vector fields. They use visual feedback of a marker at the target to estimate the relative distance and compute a speed vector field to follow a course to touchdown. The authors present a one-of-a-kind UAV landing approach based on artificial neural networks (ANNs) [14]. There, the version of education is based on fuzzy good judgment, which uses the visible detection of a marker for distance feedback.

Xuan-Munget et al. (2020) present a method for quadrotors’
self-sustaining touchdown in a moving goal, complete with a robust control technique to reduce the floor effect and other disruptions. In addition, they advocate a nation estimation based entirely on visual detection of the touchdown platform and use this estimation to plan a course to land. De Santana et al. (2019) also provide a visible-based touchdown approach for transferring objectives, tracking goal movement, and predicting a factor to land. Still, shifting goals aren’t typical in transportation jobs, and we haven’t addressed them yet. Obstacle avoidance appears to be the main issue studied in several pieces of literature on self-sufficient drone navigation [4]. This problem, however, is rarely addressed in works related to cargo transportation and touchdown. Falanga et al. (2020) present a way for dynamic impediment avoidance in the usage of occasion cameras for instant reactions of quadrotors. Experiments display the potential for the avoidance of multiple boundaries with speeds of up to ten m/s. This approach generates manipulation commands based on the detection of events during flight, and it can be used in the delivery project. Nonetheless, it focuses on drone navigation with obstacle avoidance. The authors in Chiella et al. (2019) present a localization method and a primary vector field-based method for route following in sparse forests. They use GNSS and LiDAR data for location development and the detection of bushes, which prevent the use of a probabilistic planner at the side of the vector field. Their collision avoidance strategy can also be incorporated into our solution, given that we also use a similar vector field-based management [4] technique. Despite the necessity of considering obstacles in drone navigation requirements, the issue isn’t often addressed in drone transport, transportation, and landing research because environments are typically known and controlled.

**REFERENCES**

[1] AJOT (2021) Online food delivery market to hit $ 151.5b in revenue and 1.6b users in 2021, a 10% jump in a year. https://bit.ly/6XP0z