Pathplanning Simulation of Hector Quadrotor through waypoints using PID in GAZEBO

Rohit Dey
MS, Mechanical Engineering
University of Cincinnati
Cincinnati, USA
deyrt@mail.uc.edu

Karra Sailendra
MS, Mechanical Engineering
University of Cincinnati
Cincinnati, USA
karrasa@mail.uc.edu

Oyindamola Omotuyi
Ph.D, Mechanical Engineering
University of Cincinnati
Cincinnati, USA
omotuyoa@mail.uc.edu

Twinkle Patel
MS, Aerospace Engineering
University of Cincinnati
Cincinnati, USA
pateltn@mail.uc.edu

Meher Avinash Peketi
MEng, Mechanical Engineering
University of Cincinnati
Cincinnati, USA
peketimh@mail.uc.edu

Abstract—**The Aim of the project is to control hector quadrotor to go through the waypoints by using PID(Proportional Integral and Differential) controllers using python code in mid-fidelity physics simulation software, GAZEBO. Communication between these is achieved through ROS (Robot Operating System).**

I) INTRODUCTION

In this project navigation of hector quadrotor is simulated in gazebo to make it go through waypoints. Initially Unmanned Aerial Vehicle (UAV) need to take off from its initial position and hover along the waypoints and land to the end. robot operating system (ROS) is used to publish and subscribe messages to gazebo and from gazebo. PID control is coded in python which can communicate with ROS. Default controllers in Hector quadrotor is not used and PID controller is developed in Python.

II) SOFTWARE PACKAGES

A) ROS

Robot operating system is a collection of programs which allow a user to easily control the mobile operations of a robot. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post and multiplex sensor data, control, state, planning, actuator, and other messages. Despite the importance of reactivity and low latency in robot control, ROS itself is not a real-time OS (RTOS). [1]A robot operating system (ROS) is similar to that of an operating system on a personal computer, in that it comprises a collection of programs which offer control to a user. In the case of an ROS however, these programs allow a user to control the mobile operations of a robot rather than applications on a computer. A good ROS will also make this control user-friendly. The majority of packages are licensed under a variety of open source licenses. These other packages implement commonly used functionality and applications such as hardware drivers, robot models, datatypes, planning, perception, simultaneous localization and mapping, simulation tools, and other algorithms.

ROS's core functionality is augmented by a variety of tools which allow developers to visualize and record data, easily navigate the ROS package structures, and create scripts automating complex configuration and setup processes. The addition of these tools greatly increases the capabilities of systems using ROS by simplifying and providing solutions to several common robotics development.

B) GAZEBO

Robot simulation is an essential tool in every roboticist's toolbox. A well-designed simulator makes it possible to rapidly test algorithms, design robots, perform regression testing, and train AI system using realistic scenarios. Gazebo offers the ability to accurately and efficiently simulate populations of robots in complex indoor and outdoor environments. Gazebo is a robust physics engine with high-quality graphics, and convenient programmatic and graphical interfaces. Gazebo is free and this open-source is licensed under Apache 2.0. The concept of a high-fidelity simulator stemmed from the need to simulate robots in outdoor environments under various conditions. As a complementary simulator to Stage, the name Gazebo was chosen as the closest structure to an outdoor stage. The feature of Gazebo include *Dynamics Simulation* by providing access to multiple high-performance physics engines including ODE, Bullet and DART; *Advanced 3D Graphics* by realistic rendering of environments including high-quality lighting, shadows, and textures.; *generating sensor data with noise* from laser range finders, 2D/3D cameras, kinect style sensors and contact sensors; *developing custom plugins* for robot, sensor, and environmental control which provide direct access to Gazebo's API; *creating new robot model or access in-built robot models* like PR2, Pioneer2 DX, iRobot Create, and TurtleBot and *running simulation on remote servers.*

C) HECTOR QUADROTER

A simulated quadrotor UAV for the ROS Gazebo environment has been developed by the Team Hector Darmstadt of Technische Universität Darmstadt. This quadrotor, called Hector Quadrotor, is enclosed in the hector quadrotor metapackage. This metapackage contains the URDF description for the quadrotor UAV, its flight controllers, and launch files for running the quadrotor simulation in Gazebo. Advanced use of the Hector Quadrotor simulation allows the user to record sensor data such as Lidar, depth camera, and so on. The quadrotor simulation can also be used to test flight algorithms and control approaches in simulation. The dynamics equations

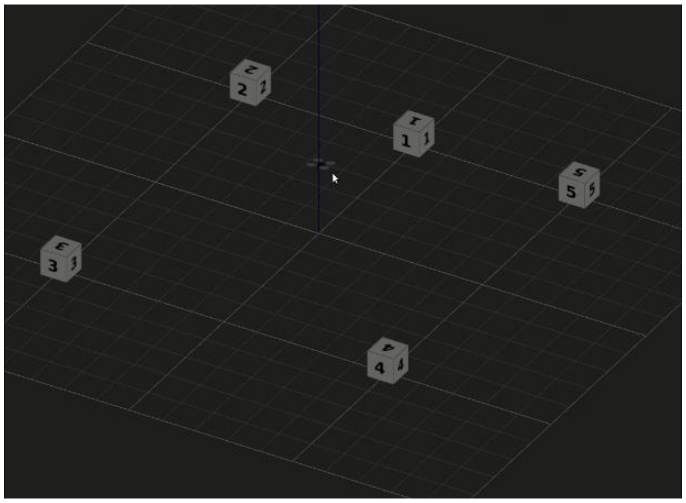

Figure I: Hector quadrotor in gazebo going through waypoints

III) METHODOLOGY

Quadrotor is spawned in to the empty world at point (0,0,0). A python program is written which can subscribe and publish required topics through ROS. A PID controller is developed for controlling different errors. Initially position of quadrotor is subscribed through topic ground_truth_pose. Through this topic we can get current x, y, z position and quaternion angle of the quadrotor[2]. These quaternions are converted to roll, pitch, yaw in every timestep. From this we calculate distance error using desired x position and desired y position, angular error from desired heading and height error from desired z position. From these three errors we can control 3D position of the quadrotor. From these three errors command linear velocity in x direction, command angular velocity in z direction, and command linear velocity in z direction using three different PID controllers. These command velocities are default topics of hector quadrotor. These topics are published to hector quadrotor using gazebo. Cmd_vel topic is used for this purpose, it contains 6 controlling parameters linear x velocity, linear y velocity, linear z velocity, angular x velocity, angular y velocity, angular z velocity. Five waypoints are fed to the system and if distance is less than one between the current position and waypoint then quadrotor switches to next waypoint. Final waypoint is on ground where it will land[3].

IV) PID

The input to the first PID Controller is the Error in distance calculated from x and y directions. The output from the PID Controller is the linear x velocity, which is published through topic cmd_x_velocity.

Error is the difference between the desired x and y position and the actual x and y position i.e., $e = d_d - d$. The PID Algorithm works in a loop. The differential error is calculated as the difference between the current error and the error from the preceding loop.

{e_dot = e – old_e}

The new integral component of error is the sum of current error and integral component of error from the previous loop.

$$\{e_{total} = e_{total} + e\}$$

As stated, linear x velocity is the output from the controller and is depicted as linear x velocity = PID (distance error). The control signal in PID is calculated as:

$$U1 = K_p * e + K_d * e\_dot + K_I * e_{total}$$

U () is the Control Signal function,
K_p is the Proportional gain = 0.3
K_d is the Differential gain = 0.05
K_I is the Integral gain = 0.001

Similarly, second PID is applied for error in angular heading in z direction. PID works in same way as mentioned for before. Error here is the difference in desired heading and current heading. It controls the topic cmd_angular_velocity_z. Gains used for this PID controller is

Kp=3; Ki=0.01; Kd=0.1

Similarly, third PID is applied for error in height in z direction. PID works in same way as mentioned for before. Error here is the difference in desired height and current height. It controls the topic command z velocity. Gains used for this PID controller is

Kp= 0.2; Ki=0.0001; Kd= 0.05.

V) Plots

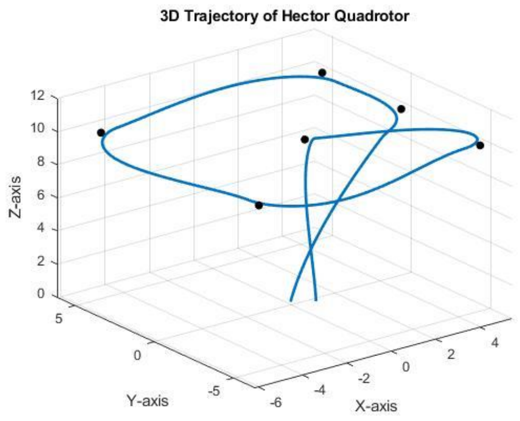

Figure:2 3D Trajectory of Hector Quadrotor

Plot is 3D trajectory of Hector quadrotor. Black points on the graph are waypoints. It starts at origin and takes off vertically which is right above the initial waypoint. Later it goes to all the points and lands.

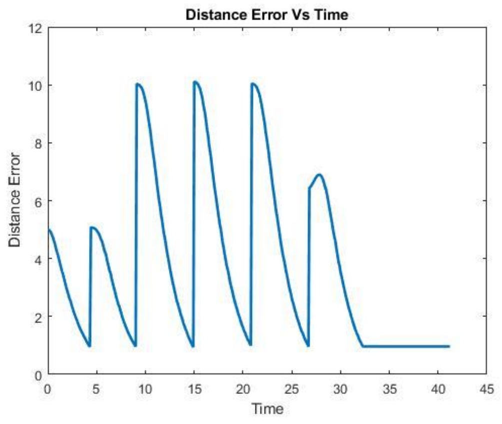

Figure:3 Distance Error

This is error in distance plot, it tries to go to zero when ever it is having a positive error using PID control. When distance is less than 1 then quadrotor goes to next waypoint, this is observed in graph. After 32 seconds quadcopter landed and no further waypoints are fed.

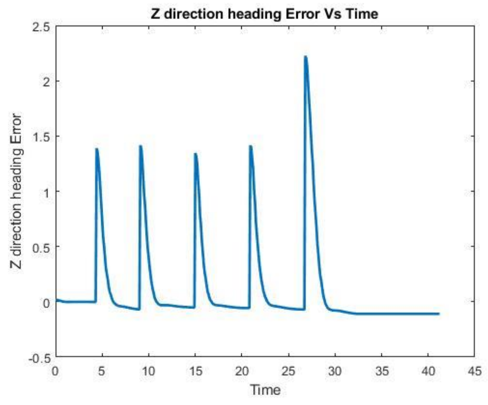

Figure: 4 Heading Errror

This plot contains heading error values. Spikes are due to sudden changes in error value to change in waypoint. It tries to go to zero.

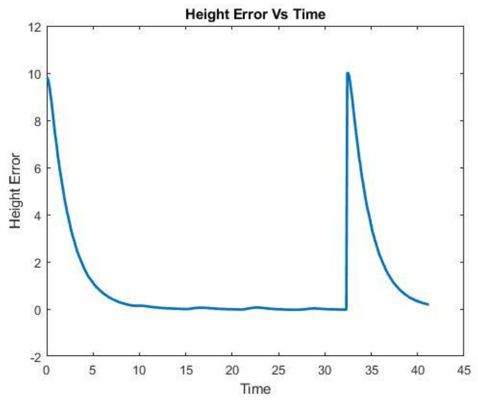

Figure: 5 Height Error

This graph shows the third control variable, height of quadrotor. Initially it takes off and tries to maintain the height. Later when it should land error increases.

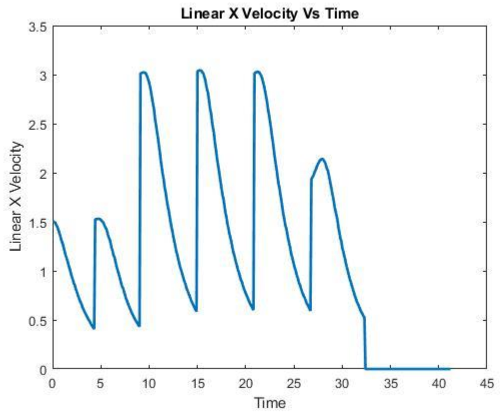

Figure:6 Linear Velocity in X direction

This linear X velocity is controlled by PID of distance error. When error is high then velocity increases to decrease the error.

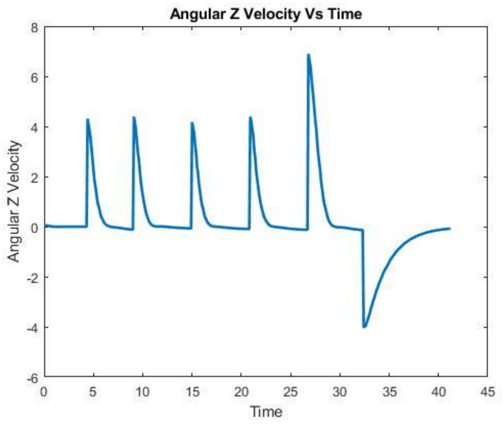

Figure:7 Angular Velocity in Z direction

This graph is angular velocity in z direction obtained by PID controller of heading error.

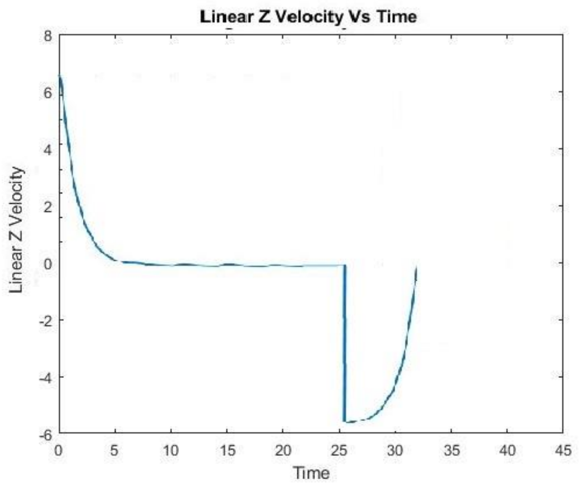

Figure:8 Linear Velocity in Z direction

This graph shows linear z velocity that is published to the rotor from PID of height error. It decreases as it is attaining the required height and it increases in opposite direction.

VI) CHALLENGES

Obtaining of 3D position control of quadrotor

Obtaining 3D control of quadrotor is a difficult thing. Model is developed which controls position in z direction, heading in z direction and x direction velocity. By this we can go any point in 3D position.

VII) CONCLUSION

Hector quadrotor is simulated in Gazebo to navigate through the required waypoints. Control of it is obtained through PID control of three different errors. 3D trajectory shows the quadrotor reaching the waypoints. Video of entire simulation is also submitted.

VIII) REFERENCES

[1] Robot operating systems: Bridging the gap between human and robot. (https://ieeexplore.ieee.org/document/6195127)

[2] https://www.researchgate.net/file.PostFileLoader.html?id=576d16ed93553b24b5721a9a&assetKey=AS%3A376462596165634%401466767085787

[3] https://scholarsarchive.byu.edu/cgi/viewcontent.cgi?article=2324&context=facpub