

Exploratory Design and Control of an Over-Actuated Drone

Riddhi Bagadiaa



Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2023-217

<http://www2.eecs.berkeley.edu/Pubs/TechRpts/2023/EECS-2023-217.html>

August 11, 2023

Copyright © 2023, by the author(s).
All rights reserved.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission.

Acknowledgement

I would like to thank my research advisor, Professor Shankar Sastry, for his guidance on this work and support in my academic career. This work would not have been possible without the efforts of Daniel Bostwick, Adith Sundaram, and Eric Berndt, who are equal partners in the authorship of this research. Special mention to Chams Mballo for his invaluable mentorship. Thank you to Professor Sanjit Seshia for being my second reader and providing insightful and helpful feedback. I would also like to recognize DARPA for providing us with the resources to make this possible. Most importantly, thank you to my parents, Ksshiraja and Jaeyun Seo for their support and encouragement without which none of this would have been possible.

Exploratory Design and Control of an Over-Actuated Drone

Riddhi Bagadia

Research Project

Submitted to the Department of Electrical Engineering and Computer Sciences,
University of California at Berkeley, in partial satisfaction of the requirements for
the degree of **Master of Science, Plan II.**

Approval for the Report and Comprehensive Examination:

Committee

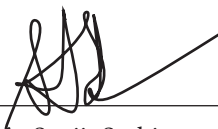


Shankar Sastry
Research Advisor

Aug. 10, 2023

(Date)

★ ★ ★ ★ ★ ★ ★



Sanjit Seshia
Second Reader

8/11/2023

(Date)

Abstract

This report presents a comprehensive study on the design and control of an overactuated quadcopter in simulation and hardware. The vehicle is engineered to autonomously maneuver intricate flight paths with better performance than a traditional quadcopter. To accomplish this, we designed a novel hardware system consisting of four thrust motors and four tilt motors, corresponding to each of the drone's four arms. This setup allows the drone to change the direction of motion more smoothly while minimizing its roll, pitch, and yaw, leading to more precise control in narrower spaces.

We developed a custom controller using the Model Predictive Control (MPC) algorithm. The MPC controller generates angular velocities for the drone's brushless motors, and tilt angles for the servo motors based on an input trajectory. The algorithm enables real-time optimization of control inputs, allowing the drone to adapt to a dynamic environment robustly. This algorithm was tested in simulation with a trajectory that mimicked a real-life slalom course on a traditional quadcopter and our drone. Keeping all other variables the same, the results of this experimental evaluation highlighted the superiority of the overactuated system in terms of maneuverability, stability, and trajectory tracking accuracy.

Our research also delves into the detailed design aspects of the overactuated UAV, including the body design, selection of appropriate materials, propulsion mechanisms, and control electronics. The hardware design was tested with the onboard MPC controller, making it perform consistent hover with real-time state feedback.

In conclusion, this research presents a novel overactuated UAV with the entire software and hardware pipeline designed from the ground up by us. The presented findings contribute to the advancement of UAV technologies, with potential applications in surveillance, search and rescue, and precision agriculture, among others. This work serves as a valuable reference for researchers and engineers seeking to enhance UAV maneuverability and control in autonomous flight scenarios.

Acknowledgement

I would like to thank my research advisor, Professor Shankar Sastry, for his guidance on this work and support in my academic career. This work would not have been possible without the efforts of Daniel Bostwick, Adith Sundaram, and Eric Berndt, who are equal partners in the authorship of this research. Special mention to Chams Mballo for his invaluable mentorship. Thank you to Professor Sanjit Seshia for being my second reader and providing insightful and helpful feedback. I would also like to recognize DARPA for providing us with the resources to make this possible. Most importantly, thank you to my parents, Ksshiraja and Jaeyun Seo for their support and encouragement without which none of this would have been possible.

Contents

1	Introduction	7
2	Related Work	7
3	Hardware Setup	8
3.1	Drone Design	8
3.2	Power Distribution Model	10
3.3	Electronics	11
4	Software Setup	14
4.1	System Design	14
4.2	Communication	16
5	Model Predictive Control	17
5.1	State Model	18
5.2	Dynamics of Tilt-Rotor and Quad-Rotor	20
5.3	MPC Integration Constraint	22
5.4	Objective Function	23
5.5	Control Algorithm	24
6	Experiments and Results	28
6.1	Simulation	28
6.2	Hardware	31
7	Future Work	34
8	Conclusion	35
	Bibliography	36

List of Figures

1	3D printed arm joints of drone	8
2	Top View of drone	9
3	Preliminary design of drone	9
4	Voltage and current draw diagram from two 3S LiPo batteries	10
5	(Left): Motor thrust test to find thrust coefficient, C_T . (Middle): Motor torque test to find torque coefficient, C_Q . (Right): Motor microsecond test to find microsecond coefficient, C_M	12
6	Test stand setup	13
7	Drone Pipeline [3]	15
8	Velocity vs time graph of a tiltrotor attempting to hover with external random disturbances	29
9	Left: Traditional quadcopter traversing a slalom course; Right: Tiltcopter traversing the same slalom course; Top: Top view of the drone navigating around obstacles; Bottom: Velocity vs time graph of drones navigating through the course	30
10	Left: Thrust(N) graph of four brushless motors tested without propellers vs timesteps; Right: Orientation (radians) of the drone vs timesteps; Running MPC to make the drone hover	31
11	Applying thrust via wire connection to test lift capabilities	32
12	End-to-end test of autonomous drone	33
13	Left: Thrust(N) graph of four brushless motors during untethered flight vs timesteps; Right: Orientation (radians) of the drone vs timesteps	34

List of Tables

1	Maximum current drawn by each component in circuit 1	11
2	Maximum current drawn by each component in circuit 2	11
3	Variables used in the MPC algorithm	25

1 Introduction

In recent years, the demand for Unmanned Aerial Vehicles (UAVs) has surged due to their potential applications in various industries. However, achieving optimal performance and precise control for UAVs remains an unexplored and challenging task, especially when navigating through dense spaces and complex trajectories. Drones have proven to be effective and versatile machines due to their ability to reach remote locations that may be impossible to get to by land. These drones have many use cases from delivering packages to military and recreational use; however, current quadcopter designs are constrained to strictly using roll, pitch, and yaw to maneuver the drone, restricting the drone's movement in the horizontal plane. The dynamics and design of a traditional drone prohibit its ability to turn quickly at high speeds. We propose a fix to this issue by implementing an over-actuated UAV that can take more aggressive turns at higher speeds, thus traveling faster and more efficiently. Recent works in over-actuated systems have shown that simple PID controllers and remote-controlled drones are very efficient at navigating complex trajectories at higher speeds [5]. In this paper, we expand on these systems and present an over-actuated autonomous drone using Model Predictive Control (MPC) to guide itself through a slalom course at high speeds. We compared the results by running our MPC algorithm through a slalom course with the dynamics of a traditional quadcopter in the same environment, to observe the performance advantages achieved by the addition of tilt rotors.

We also test the algorithm on real hardware. In order to have a novel drone, we designed, wired, and built the drone completely from ground up. The MPC controller is integrated with the hardware and we test it by making the drone hover stably. We elaborate on these tests in this paper.

2 Related Work

The end goal of our project was to develop a functional full-stack hardware, control, and communication system for a tilt-rotor quadrotor. Each aspect of this stack was developed in parallel, and we took inspiration from several papers in our design process. In designing our physical drone, we took reference from several papers demonstrating design principles for traditional quadcopters and over-actuated models [2] similar to ours. These papers were used for design principle inspiration such as techniques to locate the center of mass, mount motors, and methods of mounting servos under possible shear stresses. In our control design, we use Euler angle representations for our state vector, as presented in the following reference [4]. This paper describes MPC control design

for a fixed-axis quadrotor, which we used as a starting point for parameter tuning of our own controller. Note, the above reference had a typo in its body rate/euler-angle representations, which we corrected according to an article on quaternions [8].

3 Hardware Setup

3.1 Drone Design

To create a drone for agility at high speeds we had to think about the size of the drone we wanted to use for this specific purpose. Our original design was built upon the overall skeleton of the Holybro S500 V2+ Pixhawk 4. However, for our implementation, we needed a more versatile frame with a less restricting flight control module. To achieve this design, we decided that 3D printed parts were a more viable option that allowed for more flexibility in design choice and placement of actuators and electronics. All parts were 3D printed with PETG filament on a Prusa i3 MK3S+.

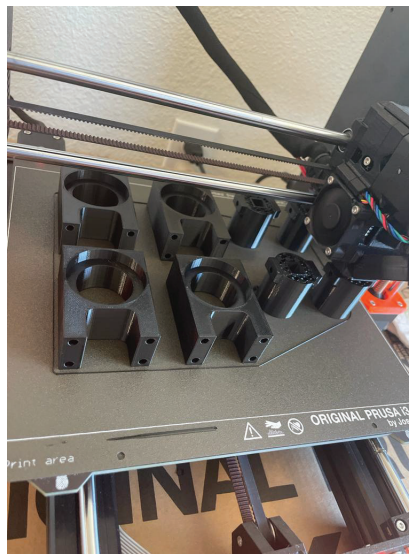


Figure 1: 3D printed arm joints of drone

We decided on an octagonal base plate with a radius of 3.5 inches. This was the smallest our drone could be in width to allow for the inner (tilt) servos and other electronic parts to fit on the drone without significant issues regarding wiring and safely harnessing flight-critical parts. To leverage this small body design, the drone is split into three levels. The base level houses the servos, the servo driver, and the breadboard that grounds all of the ESCs. The second level houses

the Arduino and the power distribution board, the third level houses the Jetson Orin Nano, and underneath the first level are the two LiPo batteries and the Realsense camera.

To properly and securely house the servos to stand upright, we devised a housing that keeps the servo upright in the vertical direction. The housing assembly that holds the carbon fiber rod end has a ball bearing on the inside that smooths the swiveling of each arm.

The camera is set up at a 58° angle to the vertical of the drone. We do this to optimize space on the drone and we take care of the resulting transformations in our calculations.

Fig. 2 shows the top view of our completed drone with the Jetson Orin Nano, bladeless motors, and the completed wiring.

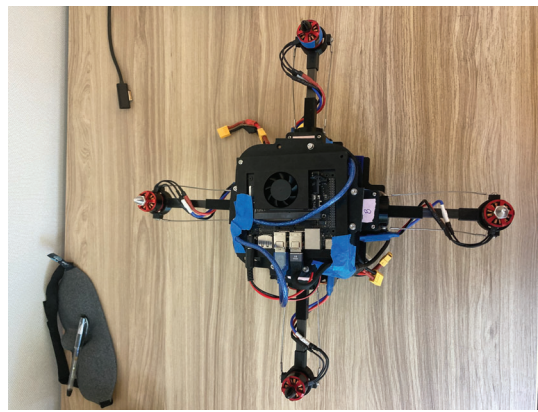


Figure 2: Top View of drone

Fig. 3 shows the CAD model of the preliminary design of our drone with just two levels instead of three. We decided to add the third layer to balance out the weight of the drone and add more stability.



Figure 3: Preliminary design of drone

3.2 Power Distribution Model

The entire drone is powered by two 3S LiPo batteries. Each battery has 2200mAh , can hold charge up to 35C , and can supply a voltage of 11.1V . Therefore, the maximum current draw supported by each battery is

$$2200 * 35 = 77000\text{mA at } 11.1\text{V}.$$

Each embedded system and actuator has a different current and voltage draw rating. In order to supply them with the accurate amount of current and voltage using the least number of batteries, we decided to use a power distribution board (PDB). Fig4 shows the power distribution circuit onboard.

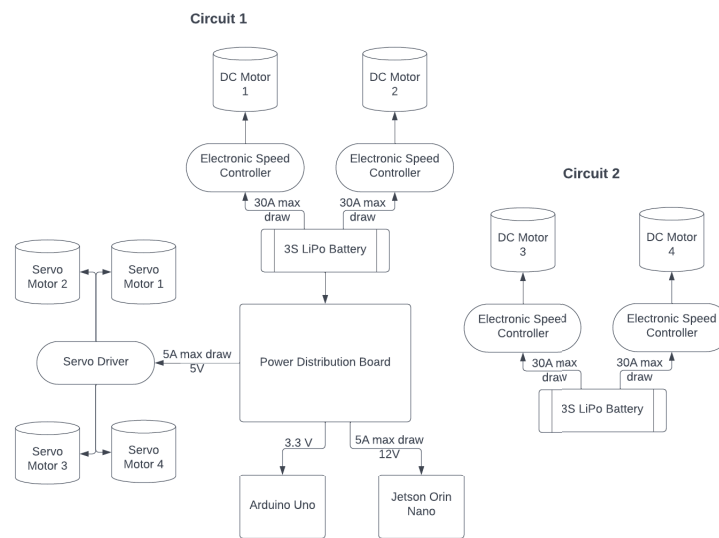


Figure 4: Voltage and current draw diagram from two 3S LiPo batteries

A power distribution board has different pins that supply different voltages. One LiPo battery powers the PDB which in turn provides power to the Arduino and Jetson Orin Nano. The Nano can take anywhere between 9 and 19 volts. Due to the limitations of the PDB, we provide the Nano with 12 volts which is sufficient to run all desired processes. The Arduino is connected to the 3.3 volt pin. The servo driver takes 5 volts and provides that to all the servo motors that are connected in parallel. The same LiPo battery also powers two brushless DC motors through the ESC.

Component	Max current draw (A)
ESC 1	30
ESC 2	30
Servo Driver	5
Jetson Orin Nano	5
Arduino Uno	0.2
Total	70.2

Table 1: Maximum current drawn by each component in circuit 1

Component	Max current draw (A)
ESC 3	30
ESC 4	30
Total	60

Table 2: Maximum current drawn by each component in circuit 2

The other LiPo battery only powers two Brushless motors. Table 1 and 2 show the maximum current that each component in the respective circuits can draw. Each ESC can draw a maximum of 30A. However, our brushless motors require at most 18.3A to provide 10N of thrust, giving an RPM of 99800. Our thrust constraint is 10N, keeping well within the limits of the ESC. The maximum current rating of the battery is 77A. Therefore, we know that we can use the 3S LiPo batteries safely.

Based on the maximum current drawn, we also had to size the wires so that they wouldn't burn. For the connection between the ESC and Lipo, we use 8 gauge wires [6] which can safely conduct 73 A of current, and for the connection between the LiPo and PDB, we use 20 gauge wires.

3.3 Electronics

Electronic Speed Controllers

The Arduino is connected to the brushless motors via an Electronic speed controller (ESC). We have four ESCs on our drone, each one is connected to a brushless motor. The ESC 30A is used to regulate and control the speed of these motors by converting the incoming electrical signals from the Arduino into appropriate voltage and current outputs that control the motor's speed and direction. The ESC is connected to the battery source as well the digital pin of the Arduino. The

power draw of an ESC 30A depends on the motors it is connected to and has a maximum power draw of 30 amps.

Brushless Motors

We use four brushless motors to provide lift to the drone. The two motors rotating clockwise are placed across from each other and the two motors rotating anticlockwise are placed on the other two mounts. We tested motors with different specifications to find one that suited our needs. The main deciding factor was the rounds per minute (RPM) per volt rating of the motors (Kv). Since our drone houses eight motors and several processing boards, it requires a relatively large amount of thrust to make it fly. Large thrust corresponds to a higher Kv . However, high Kv reduces efficiency and needs to be powered by batteries with higher voltage, aka heavier batteries. We needed motors that would adhere to our battery restrictions and require a similar amount of voltage as the other electronic parts on the drone. Additionally, higher Kv makes the motors more sensitive to disturbances and throttle changes, reducing stability. Lastly, it is advisable to use motors with lower Kv for heavier drones to obtain larger torques with lower RPM [1]. Because of the size of our drone, we wanted motors that would be more stable and less sensitive to unwanted disturbances. Based on this, we chose to go with $920Kv$ motors. $920Kv$ is in the lower range but provides enough thrust to lift our drone in the air. We also have a constraint on the maximum thrust that can be outputted by the motor at $10N$.

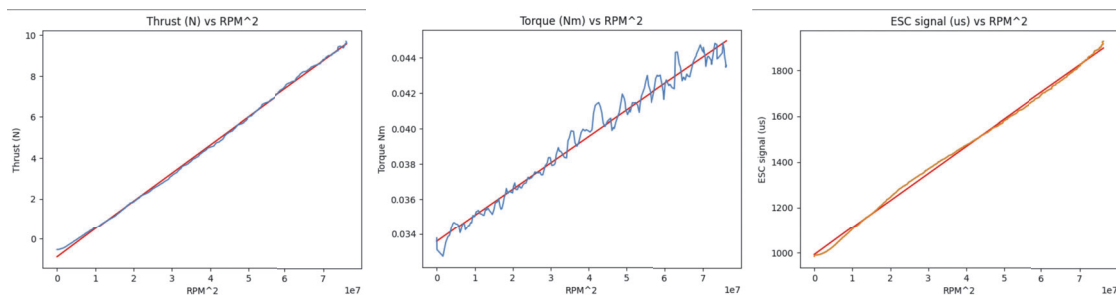


Figure 5: (Left): Motor thrust test to find thrust coefficient, C_T .
(Middle): Motor torque test to find torque coefficient, C_Q .
(Right): Motor microsecond test to find microsecond coefficient, C_M .

To get precise values of thrusts and torques for our MPC calculation we needed to find the true corresponding thrust and torque coefficients of each motor. We used the RC Benchmark testing kit to test the thrust and torque by running the motors and saving their data into a CSV file. We

then ran a linear regression on *thrust vs RPM²* and *torque vs RPM²* to give us the line that best fits our data. This line of best fit gives us the slope which is our thrust and torque coefficients.

For the actuation of the brushless motors, the Arduino receives a thrust (N) from the Jetson and runs a computational function that converts that thrust to a PWM signal in microseconds (μs) and sends it to the motors. In this calculation, we use the thrust coefficient, C_T , and the input thrust to set up a mapping of RPM^2 to μ . The power band of our brushless motors (RPM vs. PWM) is an exponential curve that is impossible to map directly to μ . Therefore, to complete the mapping, we plotted the RPM^2 vs. PWM and ran a linear regression on this data to give us another line of best fit to define our desired RPM^2 value used in the conversion. To convert thrusts to microseconds we used the following formulas in our function call:

$$RPM^2 = \frac{thrust}{C_T} \quad (.1)$$
$$\mu s = C_M \cdot RPM^2 + y_{CM}$$

The thrust variable is the input to the system, C_T is equal to $1.37456645e^{-07}$, C_M is equal to $1.19113068e^{-05}$, and the y-intercept, y_{CM} , is equal to 991.91930173.



Figure 6: Test stand setup

Fig.6 is the test stand setup to test the thrust and torque of each motor.

Propellers

The Kv of a motor also dictates the size and pitch of the propellers used. $920Kv$ motors are usually paired with propellers that have a pitch of 4.5 inches. Even though higher pitch results in more thrust, we chose propellers with a relatively lower pitch as it makes the drone more maneuverable and agile, something that is very important for our end goal. The length of the propellers was based on the size of our drone. Our goal was to install the longest propellers possible without letting it interfere with the electronics placed in the middle of the board. This is because long propellers can generate more thrust and lift, reach higher speeds, and are more efficient. We chose to go with 10 inch long propellers.

Servo Motors

We used four 20 kg (load) servo motors to over-actuate our drone, each corresponding to a specific arm. Our servos are programmed to tilt between -70° to 70° . To convert the servo angle to microseconds we created a mapping from the input angle to a corresponding microsecond value. We know that the servo can turn 180° and the full PWM duty cycle length in microseconds for our servo is between 500 and 2500. So by using the min and max of the duty cycle, we were able to map the min and max of the angle range.

4 Software Setup

4.1 System Design

We built an end-to-end system described in fig 7.

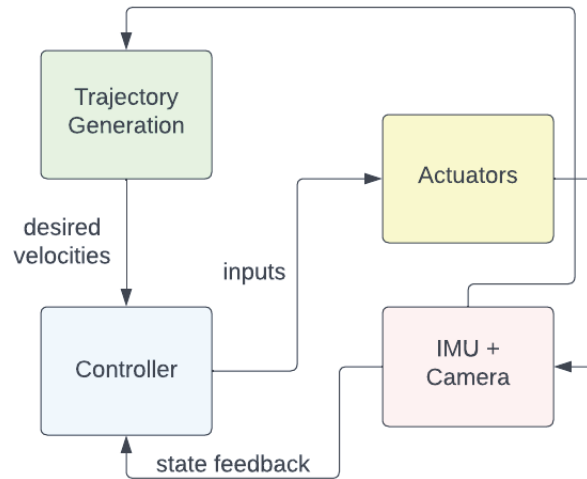


Figure 7: Drone Pipeline [3]

An Intel Realsense T265 fisheye camera located on the drone uses its visual-inertial odometry (VIO) pipeline to scan its surroundings and gather obstacle data. A localized map is created which is used by the trajectory planner to generate waypoints. Waypoints are sent to the controller which is running on a Jetson Orin Nano computing platform. The Nano runs the MPC algorithm in real-time using the state data and waypoints to produce inputs for the drone. These inputs are sent to the actuators through the Arduino Uno, making the drone fly. The new state of the drone is measured by the camera and IMU, the generated state feedback is sent back to the controller, and the model is updated.

Intel Realsense T265

We chose a tracking camera to set up our visual-inertial odometry (VIO) pipeline. The T265 has an inbuilt IMU and two fisheye cameras that capture high-resolution images of the environment.

The VIO pipeline orchestrates a sophisticated fusion of visual and inertial sensor data to enable robust motion tracking. The captured images are rectified and mapped onto a common panoramic plane and the onboard IMU sensors record precise measurements of angular velocity and linear acceleration. Using sensor fusion techniques, the pipeline synchronizes and aligns the visual and inertial data to give an estimate of the camera's pose and motion trajectory. The IMU data is capable of compensating for external accelerations and gravitational effects. The pipeline refines

pose estimates and minimizes sensor-derived errors, to output accurate and consistent position and velocity outputs that we use in our state estimation model.

We chose this RealSense camera as it has an inbuilt VIO platform that runs at approximately 200 Hz and reduces the burden of computation on the Jetson. It can also be used to calculate trajectories for reinforcement learning models.

Jetson Orin Nano Processor

The Jetson Orin Nano compute platform is composed of multiple high-performance Arm Cortex-A76 and Nvidia GPU cores, making it a suitable choice to run the MPC algorithm and other complex algorithms in real-time. We use the Orin Nano as the computing platform for all processes; it also has its own ROS2 and GPIO libraries which makes data communication very smooth. Using the onboard wifi, we are able to SSH into the board to run and perform wireless communication while testing the drone.

Arduino Uno

We have the Arduino Uno microcontroller onboard as a connection between the processor and the actuators. Using pulse width modulation (PWM) signals, we can have precise speed and direction manipulation. Furthermore, the Arduino UNO makes communication and integration seamless with serial communication protocols such as UART.

4.2 Communication

ROS2

Communication between the camera and the onboard computing platform is done via ROS2 Foxy. We set up a publisher node in the camera that publishes position and orientation data to `/camera/pose/sample` with respect to the inertial frame, and linear velocity data with respect to the body frame. The node also sends body rates data through the `/camera/gyro/sample` topic. Since our state estimation model records linear velocity information in the inertial frame, we gather information from the `/tf` topic to provide the transformation matrix between the inertial and body frame. The MPC algorithm is wrapped in a ROS2 subscriber node. The node subscribes to the topics mentioned above and parses the relevant data into a state estimation matrix, X . This matrix is used by the controller to determine the motor inputs in line `u <- mpc_controller.make_step(X)`.

Each topic may publish at different rates and may not publish all values in each message. Therefore, to ensure that there are no gaps in information sent to the controller, we use a boolean flag to ensure that values are only sent to the controller once all the data has been collected. In the meanwhile, the data that is already collected gets updated continuously so that the most recent data is sent to the controller.

Serial Communication

Serial Communication is used between the processor and the Arduino Uno to send input values, calculated by the MPC, to the motors. We use UART to establish a reliable communication link. The controller outputs four thrust values for the brushless motors and four tilt angles for the servo motors. These values are stored as a float array that is converted into a string and encoded into a byte array of fixed length for quick transmission over the UART link. The Arduino receives the byte array and converts it back into a float array to retrieve the values. Since Arduino only accepts arrays of fixed length, we ensure that the parser reads the valid bytes and discards the rest. The values are converted to pulse width modulation signals, where each signal is represented as the duration of a high pulse in microseconds. The pulse duration is sent to the Electronic Speed Controller (ESC) which conveys the information to the motors.

5 Model Predictive Control

Our high level control design is done with a closed loop Model Predictive Control(MPC) algorithm, an advanced control strategy that utilizes a mathematical model of a system to optimize control actions over a finite prediction horizon. Using the linearized system dynamics, we project the system's future behavior over n number of timesteps and compute the sequence of optimal control inputs that minimize our predefined cost function. This process is repeated at every time step, at a rate of 20 Hz, and we only apply the first set of control inputs from the sequence. n is a hyperparameter that needs to be tuned based on the desired speed and stability of the controller. The speed of the controller depends on the time taken to compute new input values with the chosen value of n and the speed of serial communication between the controller and the Arduino done at every time step. MPC allows for precise trajectory tracking and stochastic disturbance rejection when dealing with nonlinear models with complex constraints, making MPC a suitable choice for our drone. Since our goal is to test the drone in dense and complex environments, using system dynamics and future predictions helps an MPC controller surpass traditional controllers

like PID. Additionally, we optimize over a finite time frame, unlike Linear Quadratic Regulator (LQR) control which optimizes a quadratic cost function over an infinite horizon, making it more adaptable to real-world situations and capable of running in real time.

Initially, we built a position-based controller. The MPC would take in a desired position and find motor inputs to maneuver the drone to that position and orientation. However, to increase maneuverability, stability, and smoothness of control, we decided to switch to a velocity-based controller. Our controller takes in a desired velocity and produces motor values to match that velocity.

5.1 State Model

The first step is to derive the dynamics equations that represent our system and define our states and inputs. We derive our dynamics by expanding the Newton-Euler equations. Our system has 16 states, represented by X - the x , y , and z positions and velocities, the Euler angles representing the attitude of the UAV relative to our initial inertial frame, and pqr representing the body rates of the UAV. Our state representation also includes the tilt angles of the four servos.

The inputs that are sent to the eight motors are represented by u , an 8×1 vector where the first four values represent the angular velocities of our motors which directly impact the thrusts, and the next four values represent the angular velocities of our servos which change the line of action of our thrusts by rotating the arms. The choice of states and inputs was made so that our system would be control affine - i.e fit to the form $\dot{x} = f(x) + g(x)u$.

$$X = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \gamma \\ \dot{x} \\ \dot{y} \\ \dot{z} \\ p \\ q \\ r \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \beta_4 \end{bmatrix}, \quad \dot{X} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{z} \\ \dot{\phi} \\ \dot{\theta} \\ \dot{\gamma} \\ \ddot{x} \\ \ddot{y} \\ \ddot{z} \\ \dot{p} \\ \dot{q} \\ \dot{r} \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \\ \dot{\beta}_4 \end{bmatrix}, \quad u = \begin{bmatrix} \omega_1^2 \\ \omega_2^2 \\ \omega_3^2 \\ \omega_4^2 \\ \dot{\beta}_1 \\ \dot{\beta}_2 \\ \dot{\beta}_3 \\ \dot{\beta}_4 \end{bmatrix}, \quad accel_b = \begin{bmatrix} \ddot{x}_b \\ \ddot{y}_b \\ \ddot{z}_b \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} \quad (.2)$$

In (.2), $x, y, z, \phi, \theta, \gamma, \dot{x}, \dot{y}$ and \dot{z} are in the inertial frame while \dot{p}, \dot{q} and \dot{r} are body rates which are in the body frame. \dot{X} is the derivative of X .

u represents the inputs to the motors. ω_i^2 is the angular velocity squared of the i^{th} motor and $\dot{\beta}_i$ is the change in tilt of that motor.

$accel_b$ represents the linear and angular accelerations in the body frame.

We relate our body rates, $p, q,$ and $r,$ to our Euler angles by applying the following transforms:

$$\begin{aligned}
\dot{\phi} &= p + r \cos \phi \tan \theta + q \sin \phi \tan \theta \\
\dot{\theta} &= q \cos \phi - r \sin \phi \\
\dot{\gamma} &= \frac{r \cos \phi}{\cos \theta} + \frac{q \sin \phi}{\cos \theta}
\end{aligned} \quad (.3)$$

where $\dot{\phi}, \dot{\theta}$ and $\dot{\gamma}$ represent the change in roll, pitch and yaw.

5.2 Dynamics of Tilt-Rotor and Quad-Rotor

A traditional quadcopter with four motors has the following dynamics:

$$\ddot{z}_b = \frac{T_1 + T_2 + T_3 + T_4 - mg \cos \theta \cos \phi}{m} \quad (.4)$$

$$\dot{p} = \frac{(T_1 + T_2 - T_3 - T_4)l}{\sqrt{2}} + \frac{I_y qr + I_z pr}{I_x} \quad (.5)$$

$$\dot{q} = \frac{(T_1 - T_2 - T_3 + T_4)l}{\sqrt{2}} - \frac{I_x pr + I_z pr}{I_y} \quad (.6)$$

$$\dot{r} = \frac{-\tau_1 + \tau_2 - \tau_3 + \tau_4 + I_x pq - I_y pq}{I_z} \quad (.7)$$

In eq. (.5), (.6), (.7), we are calculating the body angular accelerations of the drone. In eq. (.4) we are calculating the acceleration in the z direction in the body frame. Movement in the (body frame) x and y direction is zero because thrust is only possible in the (body frame) z direction. Movement in any other spatial direction is achieved by the quadrotor flying at an angle.

I_{xx}, I_{yy}, I_{zz} are the principle moments of inertia, T_i is the thrust and τ_i is the torque produced by the i_{th} motor. l is the length of the arm.

The dynamics for a tilt copter are different. With tilt angles involved, the drone is capable of moving in the x and y direction with respect to the body frame even when roll, pitch and yaw is zero - the advantage being that we can fly in any direction while maintaining horizontal flight. The dynamics of a tilt- quadcopter are as follows:

$$\ddot{x}_b = \frac{(-T_1 \sin \beta_1 - T_2 \sin \beta_2 + T_3 \sin \beta_3 + T_4 \sin \beta_4) \cos \pi/4 - mg \sin \phi - K_x x}{m} \quad (.8)$$

$$\ddot{y}_b = \frac{(T_1 \sin \beta_1 - T_2 \sin \beta_2 - T_3 \sin \beta_3 + T_4 \sin \beta_4) \cos \pi/4 - mgsin(\theta) - K_y y}{m} \quad (.9)$$

$$\ddot{z}_b = \frac{(T_1 \cos \beta_1 + T_2 \cos \beta_2 + T_4 \cos \beta_4 + T_3 \cos \beta_3) - mg \cos \theta \cos \phi - K_z z}{m} \quad (.10)$$

$$\begin{aligned} \dot{p} = & \frac{(T_1 \cos \beta_1 + T_2 \cos \beta_2 - T_3 \cos(\beta_3) - T_4 \cos \beta_4)l}{I_x \sqrt{2}} \\ & + \frac{-\tau_1 \sin \beta_1 + \tau_2 \sin \beta_2 + \tau_3 \sin \beta_3 - \tau_4 \sin \beta_4) \cos(\pi/4) + I_y q r + I_z q r - K_p p}{I_x} \end{aligned} \quad (.11)$$

$$\begin{aligned} \dot{q} = & \frac{(T_1 \cos \beta_1 - T_2 \cos \beta_2 - T_3 \cos \beta_3 + T_4 \cos \beta_4)l}{I_y \sqrt{2}} \\ & + \frac{(-\tau_1 \sin \beta_1 - \tau_2 \sin \beta_2 + \tau_3 \sin \beta_3 + \tau_4 \sin \beta_4) \cos \pi/4 + I_x p r + I_z p r - K_q q}{I_y} \end{aligned} \quad (.12)$$

$$\begin{aligned} \dot{r} = & \frac{(T_1 \sin \beta_1 + T_2 \sin \beta_2 + T_3 \sin \beta_3 + T_4 \sin \beta_4)l}{I_z} \\ & + \frac{-\tau_1 \cos \beta_1 + \tau_2 \cos \beta_2 - \tau_3 \cos \beta_3 + \tau_4 \cos \beta_4 + I_x p q - I_y p q - K_r r}{I_z} \end{aligned} \quad (.13)$$

where $K_x, K_y, K_z, K_p, K_q, K_r$ are the respective drag coefficients.

Representing our dynamics with Euler angles results in a singularity when pitch (θ) is at 90 degrees. This is not a concern since we would not expect our drone to occupy that angle during flight, especially given the added stability of near horizontal flight made possible by the tilt-rotors.

Our inputs to the brushless motors are in RPM. However, the thrust and torque values are on a similar scale as *dtilt*. It is easier to tune a controller that optimizes for these values directly rather than RPM values. Therefore, we design our MPC algorithm to solve for thrust values, which are converted into RPM's based on eq. (.14). C_t is the coefficient of thrust and C_k is the coefficient of torque, which allows us to relate our τ to thrusts in our dynamics formulation

$$\begin{aligned} T_n &= C_t * \omega_n^2 \\ \tau_n &= C_k * \omega_n^2 \\ \tau_n &= \frac{T_n * C_k}{C_t} \end{aligned} \quad (.14)$$

5.3 MPC Integration Constraint

Next, we define the integration constraints of the MPC. using eq. (.8) to (.13). Let f^* be that equation. The nominal value of f_* is 0, representing the relationship between the state's second derivatives and the dynamics equations above. We define the following function:

$$f^* = \begin{bmatrix} x_{int}'' \\ y_{int}'' \\ z_{int}'' \\ p_{int}' \\ q_{int}' \\ r_{int}' \end{bmatrix} - \begin{bmatrix} T_{EB}(\phi, \theta, \gamma) \\ I \end{bmatrix} \begin{bmatrix} \ddot{x}_b \\ \ddot{y}_b \\ \ddot{z}_b \\ \dot{p} \\ \dot{q} \\ \dot{r} \end{bmatrix} = 0 \quad (.15)$$

In eq (.15), $[\ddot{x}_b \ \ddot{y}_b \ \ddot{z}_b \ \dot{p} \ \dot{q} \ \dot{r}]^T$ is defined by eq (.8) - (.13) in the body frame. $[x_{int}'' \ y_{int}'' \ z_{int}'' \ p_{int}' \ q_{int}' \ r_{int}']^T$ are the calculated accelerations, which drive the integration of the system states over the MPC's horizon.

In the equation above, in order to transform from the body frame to the inertial frame, we have a transformation matrix, T_{BE} , defined as

$$T_{BE} = \begin{bmatrix} c(\gamma)c(\theta) & s(\gamma)c(\theta) & -s(\theta) \\ c(\gamma)s(\theta)s(\phi) - s(\gamma)s(\phi) & s(\gamma)s(\theta)s(\phi) + c(\gamma)c(\phi) & c(\theta)s(\phi) \\ c(\gamma)s(\theta)c(\phi) + s(\gamma)s(\phi) & s(\gamma)s(\theta)c(\phi) - c(\gamma)s(\phi) & c(\theta)c(\phi) \end{bmatrix} \quad (.16)$$

where $c(\cdot)$ is a cosine function and $s(\cdot)$ is a sine function.

We left multiply $\begin{bmatrix} \ddot{x}_b & \ddot{y}_b & \ddot{z}_b & \dot{p} & \dot{q} & \dot{r} \end{bmatrix}^T$ with T_{EB} to get the transformed values in the world frame.

To be able to run our optimization in real time, we are using adaptive MPC, and linearizing our system at every time step. We linearize f^* using the Taylor Series Expansion. For this, we can find the Jacobian of each of the state variables with respect to f^* and use them to find the expansion.

Let A , B and C be the Jacobians of state variables X , u and time-varying parameter, acceleration, respectively.

$$\begin{aligned} A &= \frac{\partial f}{\partial x} \\ B &= \frac{\partial f}{\partial u} \\ C &= \frac{\partial f}{\partial \begin{bmatrix} \dot{v} \\ \dot{w} \end{bmatrix}} \end{aligned} \tag{.17}$$

Adding eq.(.17) to the Taylor Series expansion we get,

$$\delta f = o - f^* = A \delta X + B \delta u + C \delta \begin{bmatrix} \dot{v} \\ \dot{w} \end{bmatrix} \tag{.18}$$

Eq. (.18) is the Euler Lagrange function.

5.4 Objective Function

The objective function is where we set the desired behavior of our controller, including setting our reference trajectory and doing the bulk of our controller tuning. We define it using the loss term, meyer term and the r-term. The loss term is a cost function that quantifies the error associated with the state, x_k , control inputs, u_l , time-varying parameters, z_k and target pose, p , over time k . The meyer term is the terminal cost or penalty function associated with the state at the end of the prediction horizon. Here we set the meyer term equal to the loss term. Our primary objective is to

track our desired velocities (the terms with the highest gains), while our secondary objective is for stabilized horizontal flight. We do most of our tuning by adjusting gains within our objective function which is defined as a soft constraint, rather than adding additional hard constraints to our problem, which can make it unsolvable.

$$l(x_k, u_l, z_k, p) = m(x_n) = K_1 * ((\dot{x}_d - \dot{x})^2 + (\dot{y}_d - \dot{y})^2 + (\dot{y}_d - \dot{y})^2) + K_2 * ((\dot{p})^2 + (\dot{q})^2 + (\dot{r})^2) + K_3 * ((\phi)^2 + (\theta)^2 + (\gamma)^2) \quad (.19)$$

The r-term is

$$r(u_k) = \Delta u_k^T R \Delta u_k \quad (.20)$$

Putting everything together we get the objective function, C ,

$$C = \sum_{k=0}^{n-1} (l(x_k, u_l, z_k, p) + \Delta u_k^T R \Delta u_k) + m(x_n) \quad (.21)$$

5.5 Control Algorithm

Once we set up our model with all the dynamics mentioned above, we can define our constants and variables. Table 3 describes all the constant values and variables used in our algorithm.

Variable	Value	Description
g	$9.81ms^{-1}$	acceleration due to gravity
m	$2.19kg$	mass of drone
l	$0.2212m$	arm length of drone
$side_arm_len$	$l/\sqrt{2}$	arm length along the x-axis and y-axis
I_{xx}	0.00989	principle component of inertia of drone in x direction [7]
I_{yy}	0.01254	principle component of inertia of drone in y direction
I_{zz}	0.00994	principle component of inertia of drone in z direction
C_t	$1.242e^{-7}/1.242e^{-7}$	coefficient from w^2 to thrust
C_k	$1.4799e^{-10}/1.242e^{-7}$	coefficient from w^2 to torque
$K_x = K_y = K_z$	0.2	linear drag coefficients
$K_p = K_q = K_r$	0.2	moment drag coefficients
$n_horizon$	6	MPC horizon
t_step	0.04s	duration of each step

Table 3: Variables used in the MPC algorithm

$n_horizon$ and t_step are variable values that we tune based on the performance of our controller. We initially started with $n_horizon = 20$. This means that our controller is projecting 20 time steps into the future as it calculates its sequence of optimal control inputs. In this process, we are optimizing our sequence of inputs to minimize our cost function over the time horizon; this means that the shorter our horizon, the faster the controller will attempt to converge to our reference trajectory r . A controller with a shorter horizon is more reactive and will have a higher overshoot, while a controller with a larger horizon will have less overshoot but will be less reactive. A longer horizon also increases compute time, as we need to optimize over a larger sequence of inputs. We needed to find a $n_horizon$ that was able to stabilize the drone, prevent overshoot, and also be able to run in real-time without causing any delays. Thus, after trial and error, we settled for $n_horizon = 6$. t_step is the duration of each interval for which the controller is run before executing the next set of outputs. The shorter the interval, the more often the control loop runs. We started off with $t_step = 0.01s$. For $t_step = 0.01s$ and $n_horizon = 6$, the MPC controller would look ahead by 0.06 seconds and then give us the control output for 0.01 seconds. The key challenge here was that in order to accurately update the motors in real time at every timestep,

t_{step} , the MPC had to compute new values within 0.01 seconds, which was not possible on our compute platform. Therefore, based on experimentation, we decided to have $t_{step} = 0.04s$.

At every timestep, the Realsense camera VIO pipeline publishes data to the topics, `/camera/pose/sample` and `/camera/gyro/sample`. Our script subscribes to these topics and updates the state data. From `/camera/pose/sample` we get `Pose.position`, `Pose.orientation` and `Twist.linear_velocity`. From `/camera/gyro/sample` we get `angular_velocity`. Algorithm 1 describes the way data is pulled from the topics and stored in the `state_estimate`. Once the data is gathered, Algorithm 2 uses that data to update the MPC controller and find input values, u , which are sent to the motors through the Arduino using serial communication.

Algorithm 1 Getting state data from the camera

```
1: state_estimate ← []
2: pose_twist ← []           ▷ initialize arrays to store data received from topics
3: imu ← []
4:
5: function POSE_TWIST_DATA(msg)   ▷ msg stores data received from camera/pose/sample
6:   if msg ≠ NULL then
7:     pose_twist ← msg           ▷ pose_twist stores the relevant values of msg
8:   end if
9:   send_data()
10: end function
11:
12: function IMU_DATA(msg)         ▷ msg stores data received from camera/gyro/sample
13:   if msg ≠ NULL then
14:     imu ← msg                   ▷ imu stores the relevant values of msg
15:   end if
16:   send_data()
17: end function
18:
19: function SEND_DATA
20:   if imu ≠ NULL & pose_twist ≠ NULL then ▷ checks if new data has been added to imu
   and pose_twist
21:     state_estimate ← imu + pose_twist
22:     pose_twist ← []             ▷ empty arrays for new data
23:     imu ← []
24:   end if
25:   run_mpc()
26: end function
```

Algorithm 2 Running Model Predictive Control Algorithm at each timestep

```
1:  $des\_vel = desired\_velocities[0]$ 
2:  $X_t \leftarrow state\_estimate$   $\triangleright$  state_estimate data is gathered from the VIO pipeline
3:  $X_{t-1} \leftarrow X_t$ 
4:  $u \leftarrow [0, 0, 0, 0, 0, 0, 0, 0]$ 
5: function RUN_MPC
6:    $u \leftarrow mpc\_controller.step(X)$   $\triangleright$  run the controller to compute the next set of inputs based
   on the current state
7:    $to\_motors(u)$   $\triangleright$  send the computed inputs to the motors
8:    $X \leftarrow state\_estimate$   $\triangleright$  update state data
9:    $accel \leftarrow \frac{X_t - X_{t-1}}{\Delta t}$ 
10:   $mpc\_controller.X \leftarrow X_t$ 
11:   $mpc\_controller.u \leftarrow u$   $\triangleright$  update the values stored by the controller object
12:   $mpc\_controller.a \leftarrow accel$ 
13:   $X_{t-1} \leftarrow X_t$ 
14:  if  $des\_vel$  is reached then
15:     $des\_vel \leftarrow next(desired\_velocities)$   $\triangleright$  go towards the next desired velocity given by the
    trajectory
16:  end if
17: end function
```

6 Experiments and Results

6.1 Simulation

Hover with external disturbances

The first big test we conducted was to test the stability of the MPC controller by making the drone hover in the presence of external disturbances. We modeled noise using $random.normal(0, 0.01, 16)$ and added it to the state model. The results of the test are depicted in fig. 8.

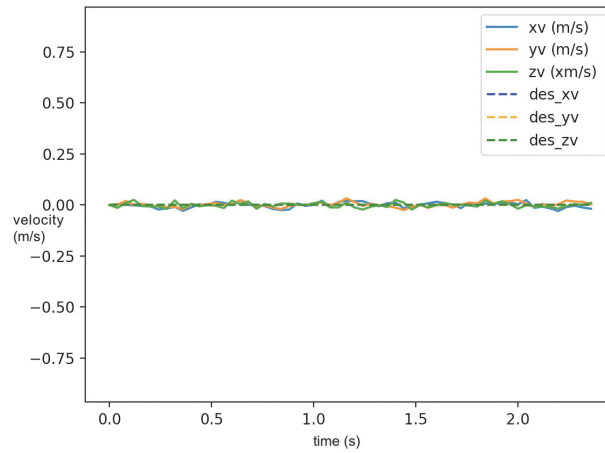


Figure 8: Velocity vs time graph of a tiltrotor attempting to hover with external random disturbances

The controller is able to interpret the disturbances and continue to maintain steady flight without changing velocity, in the air.

Slalom Course

To quantify the performance of our tilt-rotor compared to a traditional quadcopter, we ran both drones through a slalom course. Both drones were exactly the same in all aspects except one- the tiltrotor had four more servos than the quadcopter. Fig. 9 shows the results of navigating through the course.

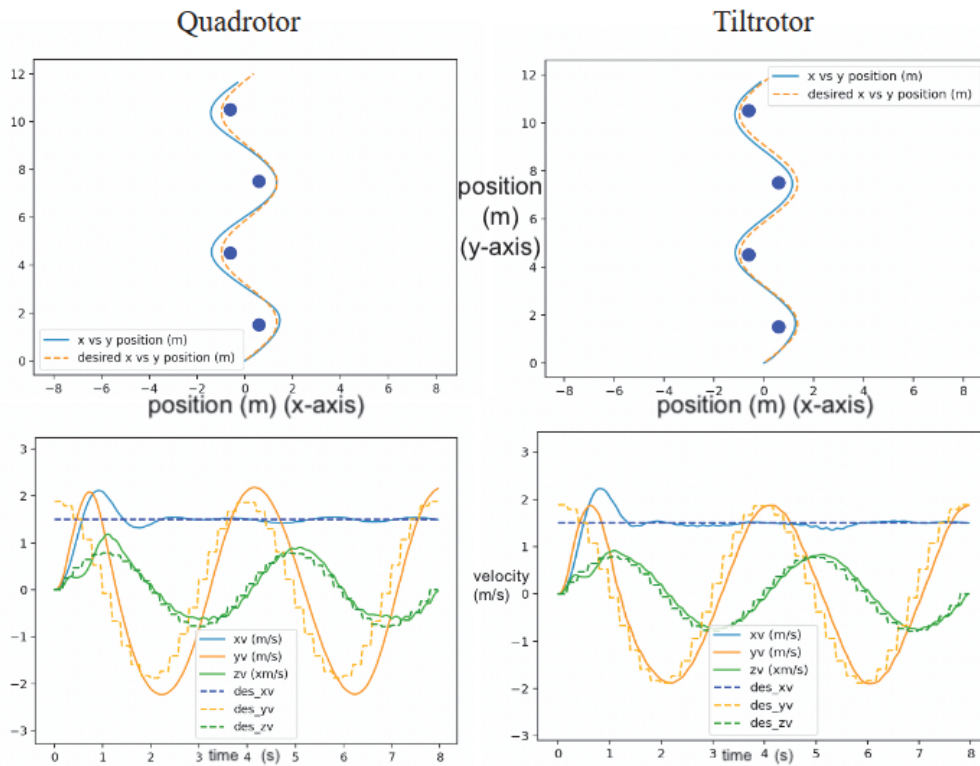


Figure 9: Left: Traditional quadcopter traversing a slalom course; Right: Tiltcopter traversing the same slalom course; Top: Top view of the drone navigating around obstacles; Bottom: Velocity vs time graph of drones navigating through the course

The two graphs on top show the top view of the obstacle course where the blue dots are the obstacles. The orange dotted line is the path determined by the MPC that both drones are trying to follow. This path is the same for both drones. The blue solid line is the drone trying to follow that path in simulation. We see that in places where the MPC expects the drone to make a tight maneuver, the traditional quadcopter overshoots and deviates noticeably from the path. The tiltrotor on the other hand follows the given trajectory more smoothly and accurately.

The two graphs at the bottom give a more clear view of the desired vs actual velocity. We see a noticeable right shift in the actual y velocity graph from the desired y velocity of the quadcopter. All the velocities of the tilt rotor, on the other hand, are more accurate. This is because having tilt motors allows the drone to instantaneously change direction without affecting velocity in the z direction.

6.2 Hardware

Testing Environment

Our hardware test environment was in Cory 391 as it had a test space separated by a safety net. We had three different test formats. We started by running the desired test on the drone without blades. This gave us output values that we could graph and analyze. Following the success of that, we placed the drone in the test space and repeated the same test, keeping the drone tethered by a wire. Lastly, we tried to make the drone hover while leaving it untethered. In fig.12 the drone is placed on a box so that the camera can be kept off the ground for calibration.

Hover test without propellers

We tested the drone without propellers by manually tilting it and watching the MPC stabilize itself. The graphs below show the thrust and RPY produced by the motors over time based on the MPC, to maintain the given desired velocity.

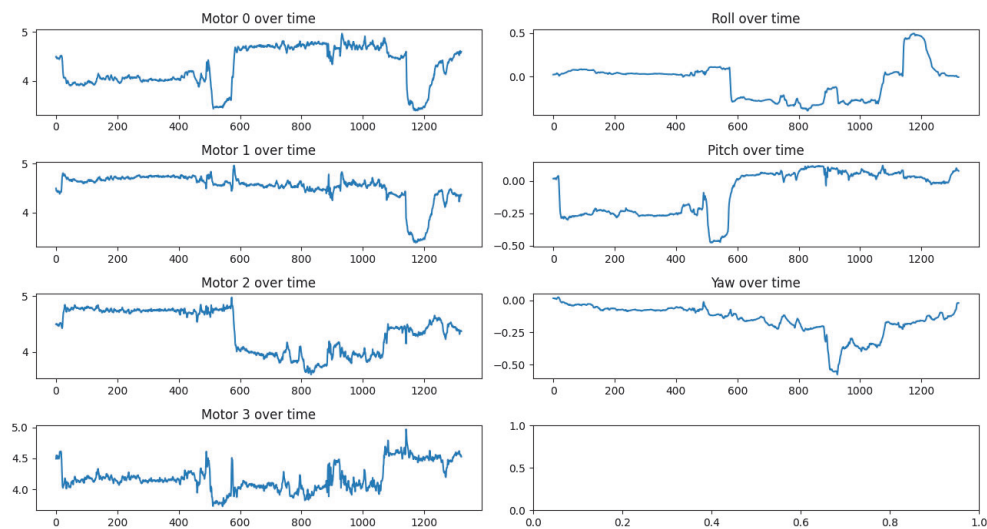


Figure 10: Left: Thrust(N) graph of four brushless motors tested without propellers vs timesteps; Right: Orientation (radians) of the drone vs timesteps; Running MPC to make the drone hover

Fig. 13 shows the effect of thrust on the roll, pitch, and yaw of the drone. For the first 600 timesteps, all four motors maintain a constant and equal thrust, thereby keeping the RPY stable as well. During the second half, we manually roll the drone to mimic imbalance. As a result, the

MPC tries to balance it out by decreasing the thrust values on motor 2 and 3, so that the drone can correct itself and become horizontal again.

Hover test with propellers

We attached propellers to our drone to test flight. At first we manually increased the thrust over multiple timesteps to watch the drone take off and hover in the air. We were successfully able to do so as shown in fig. 11.

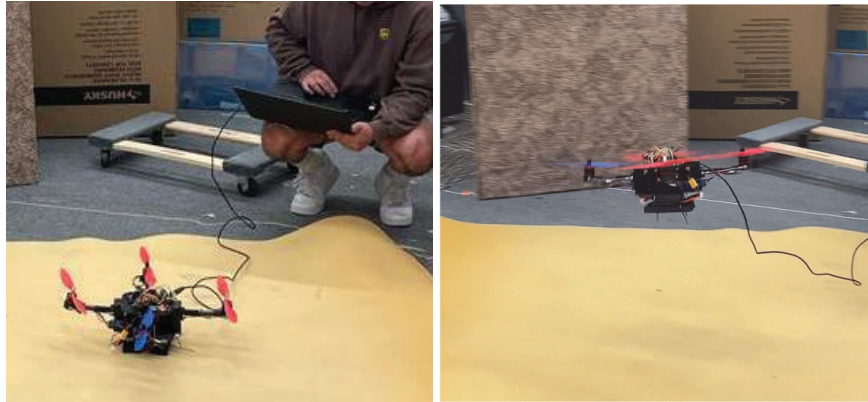


Figure 11: Applying thrust via wire connection to test lift capabilities

We then ran the controller onboard the Jetson without any tethers. The setup is shown in fig. 12.

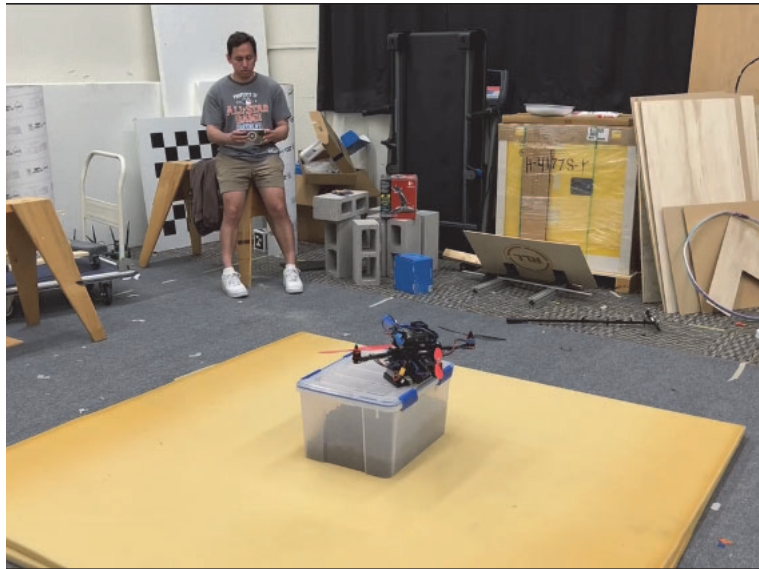


Figure 12: End-to-end test of autonomous drone

Our goal was to make it hover 0.15 meters above the ground and then increase the thrust to increase its height. Fig. 13 shows the stable flight of the drone maintaining a constant hover for 25 time steps before increasing thrust uniformly to gain height. The drone remains relatively flat without any unexpected rotations, as seen by the three graphs on the right.

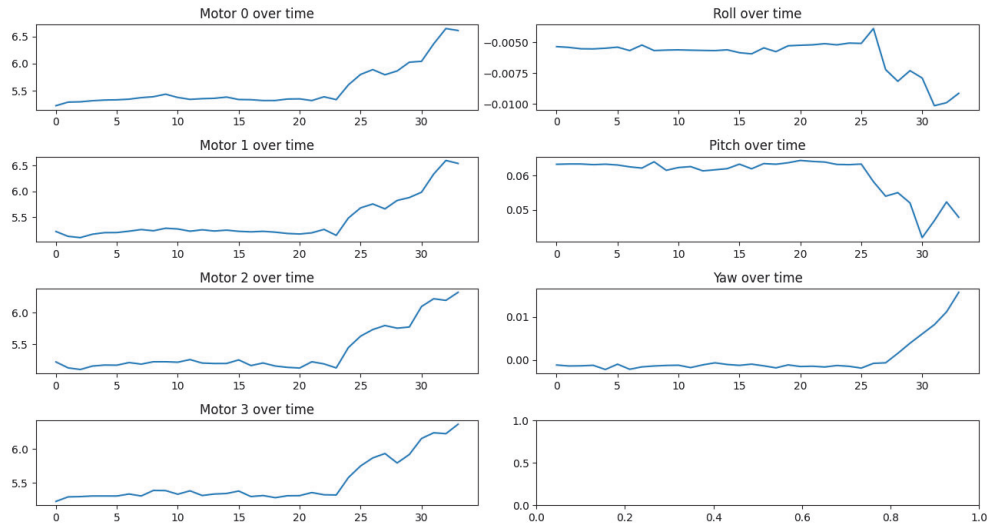


Figure 13: Left: Thrust(N) graph of four brushless motors during untethered flight vs timesteps; Right: Orientation (radians) of the drone vs timesteps

Thus, we were able to test the software set up on the drone and conduct preliminary tests with promise. While everything works smoothly in simulation, there were many challenges we faced during hardware testing which still need to be ironed out.

7 Future Work

We would like to continue working on this project and there are several ways to do so. The immediate next steps would be to run the hardware through the same slalom course and compare it with a traditional quadcopter. This is challenging because incorporating hardware introduces many more challenges such as uncertainty and non-deterministic behavior of electronics. We would also like to take this project one step further by making it a neuro-symbolic system. Currently, we are providing it with a pre-determined trajectory with the help of waypoints. The goal is to make the drone navigate through an unknown environment by determining the next target position using a Reinforcement Learning model. Using visual odometry data, the model should be able to give a suitable waypoint after learning its surroundings and avoiding obstacles. Another future work could be to do an analysis of battery charge and discharge and optimize the battery usage for more efficient flight. This could also lead to better battery choices that would help reduce the weight of the drone.

8 Conclusion

In conclusion, this technical research report presents the design, construction, and evaluation of a novel tilt rotor drone featuring eight motors. This undertaking encompassed the development of both, the hardware and software pipeline, along with the utilization of 3D printing techniques to fabricate essential parts. Our objective was to assess the drone's performance, particularly in comparison to conventional quadcopters, within various operational environments.

The integration of tilt rotor functionality proved to be a key differentiator in the drone's behavior. Simulation results, based on the slalom course tested in fig. 9, indicated that the presence of tilts increased the drone's capability to excel in challenging scenarios, such as densely populated spaces and complex trajectories. This outcome unleashes the potential of the proposed design to surpass the limitations of conventional quadcopters and be applied in scenarios that require more precision and maneuverability.

Experimentation with this tilt rotor design bears implications for broader aerial exploration. The expanding space flight industry, characterized by evolving challenges and demands, stands to benefit from the adaptability exhibited by the over actuated drones.

On the hardware front, this research was able to prove the conceptual design and translate it into functional hardware components. Acknowledging the constraints of the current motor setup, having more sophisticated motors would allow for deterministic flight and more reliable testing, thereby elevating the overall performance of the drone.

The seamless integration of the hardware and software stack was instrumental in enabling flight operations and establishing a foundation for autonomous control. Notably, we were able to verify the application of Model Predictive Control (MPC) for autonomous flight and its viability in the context of the tilt rotor drone.

In summary, this research has realized a novel tilt rotor drone design from conception to implementation. The integration of hardware design, software architecture, and fabrication techniques has yielded a platform that can be used as a test bed for future designs. The observed advantages in confined spaces indicate promising potential for further exploration in applications demanding precision flight. As the aerospace industry evolves and demand increases, the need for new directions in aerial exploration also increases. We remain hopeful that our exploratory research paves the way for continued advancements in the realm of aerial robotics.

Bibliography

1. S. R. ABDUJABAROV NURIDDIN TAKHIROV JONIBEK. *USING FLIGHT CONTROL SYSTEMS IN UNMANNED AERIAL VEHICLES*. Accessed: August 2, 2023. 2022. URL: <https://www.elibrary.ru/item.asp?id=48514583>.
2. J. B. B. Ali Bin Junaid Alejandro Diaz De Cerio Sanchez. "Design and Implementation of a Dual-Axis Tilting Quadcopter". In: *Kinematics and Robot Design I, KaRD2018* 7:4, 2018. Accessed: August 2, 2023, p. 65. URL: <https://www.mdpi.com/2218-6581/7/4/65>.
3. *Lucid Charts Website*. Accessed: August 11, 2023. URL: <https://lucid.app>.
4. M. O. M Islam and M. M. Idres. "Dynamics and control of quadcopter using linear model predictive control approach". In: *Journal of Physics: Conference Series* 270:1, 2017. Accessed: August 2, 2023, p. 012007. DOI: 10.1088/1757-899X/270/1/012007. URL: <https://iopscience.iop.org/article/10.1088/1757-899X/270/1/012007>.
5. D. B. D. Michael W. Oppenheimer and M. A. Bolender. *Control Allocation for Over-actuated Systems*. Accessed: August 2, 2023. 2023. URL: <https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4124855>.
6. PowerStream. *Wire Size Calculator*. Accessed: August 2, 2023. 2023. URL: https://www.powerstream.com/Wire_Size.htm.
7. *SOLIDWORKS Website*. Accessed: August 2, 2023. URL: <https://www.solidworks.com/>.
8. R. F. Stengel. *Lecture Notes on Quaternions*. Accessed: August 2, 2023. 2019. URL: <http://www.stengel.mycpanel.princeton.edu/Quaternions.pdf>.