Offboard MPC for Environment-aware Payload Delivery Drone

Aneesh Sinha, Balaji Praneeth Boga, Devansh Dhrafani, Vedang Kokil, Vedant Gite, Mark Bedillion
Department of Mechanical Engineering
Carnegie Mellon University
{aneeshsi, bboga, ddhrafan, vkokil, vgite, capn}@andrew.cmu.edu

***Abstract*—This work introduces an Offboard Non-linear Model Predictive Control (NMPC) framework implemented on the Crazyflie 2.1 hardware, with a comparative analysis against a baseline Proportional-Integral-Derivative (PID) controller for quantitative and qualitative assessment. The ACADOS solver is utilized for efficient C code generation, enabling the NMPC controller to operate at a frequency of 66.67Hz on the offboard base station.**

In the evaluation of performance metrics, a detailed analysis is conducted to discern the distinct capabilities of NMPC and PID controllers. Additionally, an exploration is undertaken to understand the influence of payload weight on the miniature quadcopter's performance. The findings reveal the NMPC controller's superior trajectory tracking abilities compared to its PID counterpart.

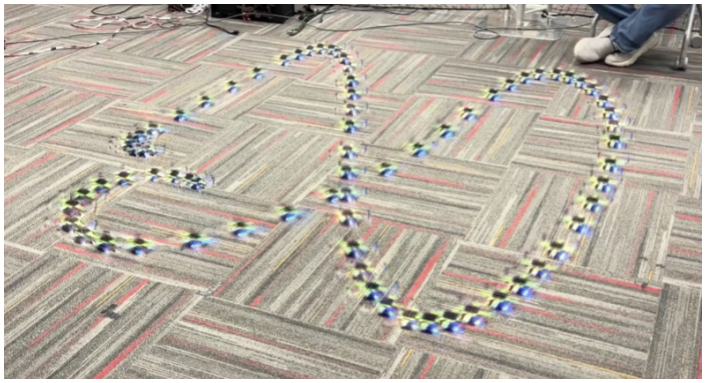

Fig. 1. Crazyflie drone aggressively tracking figure of 8 trajectory using NMPC framework

I. Problem Description

A. Background

Unmanned aerial vehicles (UAVs) have revolutionized various industries, showcasing their potential in logistics, emergency services, and more. As drones become integral to modern applications, the need for advanced capabilities in dynamic environments has become increasingly evident.

B. Problem Statement

Unmanned aerial vehicles (UAVs) have gained significant traction in various applications, particularly in the realm of package delivery. However, the existing challenges associated with dynamic environments, obstacles, and payload weight necessitate the development of a sophisticated drone delivery system. The objective of this project is to design and implement an advanced drone delivery system that can intelligently plan around obstacles to reach its destination while considering the impact of payload weight on performance. The system will be developed using a Nonlinear Model Predictive Control (NMPC) framework as the primary control architecture, and its performance will be compared with a baseline Proportional-Integral-Derivative (PID) control strategy.

C. Objectives

The objectives for the project are listed below: (1) Baseline PID Comparison: Develop a baseline Proportional-Integral-Derivative (PID) control strategy for drone delivery and compare its performance against the NMPC framework. Also, assess the strengths and limitations of both control architectures in terms of path tracking, stability, and responsiveness. (2) Intelligent Path Planning: Implement obstacle detection and avoidance mechanisms to ensure safe and efficient navigation. (3) NMPC Control Architecture: Utilize the Nonlinear Model Predictive Control (NMPC) framework as the primary control architecture for the drone delivery system. (4) Payload Weight Consideration: Investigate the impact of payload weight on the performance of both the NMPC and PID control strategies. Optimize the control algorithms to adapt to varying payload weights and ensure consistent and reliable deliveries.

D. Challenges

The challenges for the project are listed below: (1) Environmental Adaptability: Traditional drones struggle to navigate smoothly through environmental conditions, impacting overall adaptability and performance. Addressing this challenge involves developing planning and control mechanisms to enhance the drone's ability to handle varying environmental conditions effectively. (2) Precision and Stability: Previous instances of payload delivery have been marred by difficulties in achieving the required precision and stability during crucial phases, leading to sub-optimal outcomes. (3) Communication Delays through Crazyflie Radio: Develop solutions to challenges associated with communication delays through the Crazyflie radio, a crucial aspect of real-time control. (4) Inaccurate state estimates: Solve challenges related to the inaccurate results from Flowdeck measurements and try to mitigate these issues by finding reliable global/relative state

estimate. (5) Payload Design: Tackle challenges in designing a payload related to weight and ergonomic design.

II. Related literature

We first conduct a literature review to identify existing approaches for doing Model Predictive Control (MPC) on quadrotors. The goal of this literature review was to identify existing approaches, and challenges involved with the same. The MPC optimization objective can be solved either onboard using the quadrotor's embedded platform or offboard on a much powerful base station computer. The choice of where to run the solver for MPC has major consequences on the system architecture and performance.

A. Onboard MPC

If the MPC controller is run onboard, it is limited by the computational power of the quad rotor's embedded platform. Most modern quad rotor platforms [1] are designed with powerful GPU's for running large neural network models but feature slower CPU's. These embedded platforms are constrained by size and weight to be able to fit on a mobile platform like quad rotors, which have severe payload restrictions. Due to this, the CPU processing speed is limited. For example, the Jetson Nano, a popular choice of embedded computers for quad rotors, has a 1.43GHz 4-core Cortex-A57 CPU. This is much slower than modern computers which typically have CPU's with a processing speed of 2.5 GHz and 8 or more cores. Our target platform, the Crazyflie 2.1 [2], has a Cortex-M4 CPU with a max clock frequency of 168MHz. This severely limits the speed at which we can solve the optimal control problem for MPC.

Most modern works [3], [4] run the MPC controller onboard. Falanga et al [3] uses the ACADOS [5] with qpOASES [6] solver to compute trajectories at a frequency of 100Hz, with a time horizon of 2s and a discretization step of 0.1s. The solver is run on a Qualcomm Snapdragon Flight board with a quad-core ARM processor at upto 2.26 GHz and 2GB of RAM. They chose the ACADOS solver because of two main reasons: (1) ACADOS is capable of handling single and multiple shooting and integration for any transcribed system dynamics. (2) It generates fast C code which can be compiled on the target embedded platform and uses all available accelerators and optimizations tailored for that platform.

Alavilli et al TinyMPC [4] uses the alternating direction method of multipliers (ADMM) algorithm which leverages the structure of the MPC problem by precomputing and caching while avoiding divisions and matrix inversions online. This approach facilitates rapid computation and has a small memory footprint, enabling deployment onto resource-constrained MCUs. TinyMPC was tested on the Bitcraze Crazyflie 2.1 platform [2] and ran the MPC controller with linearized dynamics at 500Hz on the onboard Craziflie MCU.

B. Off-board MPC

Off-board MPC techniques are not limited by the computational capacity of like the embedded platforms as they can run on modern hardware like the latest desktop CPU's. This enables the MPC controller to run as fast as the limits of modern CPU's. But off-board control methods suffer from communications delay. To run any feedback based controller like MPC, a full state estimate must be provided. For an off-board control case, state information like position and attitude must supplied either via an onboard state estimator or from a motion capture system. This introduces an inherent communication delay corresponding to the latency of the communication channel like a radio or WiFi. The off-board MPC controller must account for this delay to enable efficient control.

Carlos et al An Efficient Real-Time NPMC for Quadrotor Position Control under Communication Time-Delay [7] uses an off board control architecture to address limited onboard computational resources and utilizes acados package for solving optimal control problem. The solution was demonstrated on the Crazyflie 2.1 nano-quadrotor and implemented a real-time iteration SQP scheme, while employing HPIPM and BLASFEO for efficient solution times.

III. System modeling

One of our objectives is to optimize the payload delivery time, making it efficient and fast. Given that most quadrotors have limited flight times which are further shortened due to additional payload weight, we want the quadrotor to move as fast as possible. We therefore decided go with the full non-linear dynamics for the Crazyflie to get the maximum performance possible. We select the state vector [s = insert state vector equation]. Here p is the position in world coordinates, $q = (q_w, q_x, q_y, q_z)^T$ is the quaternion orientation, v_b is the body velocity, and ω is the body angular velocity. The control input is [u = insert control input]. Here, Ω is the individual motor angular velocity. Equation [refer MPC objective + constraints] describes the MPC objective function and constraints.

$$
\begin{aligned}
\min_{\substack{\mathbf{x}_{0:N-1} \\ \mathbf{u}_{0:N-1}}} \sum_{k=0}^{N-1} & \left[\frac{1}{2}(x_k - x_{ref,k})^T Q (x_k - x_{ref,k}) + \frac{1}{2} u_k^T R u_k \right] \\
& + \frac{1}{2}(x_N - x_{ref,N})^T Q_f (x_N - x_{ref,N}) \\
\text{s.t} \quad & x_1 = x_{IC} \\
& x_{k+1} = f(x_k, u_k) \quad \forall \text{ k=1,2,...N-1} \\
& u_{min} \leq u_k \leq u_{max} \quad \forall \text{ k=1,2,...N-1}
\end{aligned}
\tag{1}
$$

We use the full non-linear dynamics as described in 2. Here, S is the quaternion rotation matrix from body to world frame. $\bigotimes$ is the quaternion multiplication. m is the mass of the Crazyflie, F_b is the collective body force. $J = diag(J_{xx}, J_{yy}, J_{zz})$ is the moment of inertia matrix, and M_b is the moments applied to the COM of the quadrotor. Finally, $1_z = (0, 0, 1)^T$.

$$\dot{x} = f(x,u) = \begin{bmatrix} Sv_b \\ \frac{1}{2}q \bigotimes \omega \\ \frac{1}{m}F_b - S^T g 1_z - \omega \times v_b \\ J^{-1}(M_b - \omega \times J\omega) \end{bmatrix} \quad (2)$$

IV. Controller Design

The primary objective of the project is to conduct a comparative analysis between a baseline control strategy and an optimal control strategy. The open-source hardware of the Crazyflie is equipped with default controllers that rely on PID. Consequently, the selection of the baseline controller was determined to be PID. The advanced controller chosen for comparison is MPC. The decision to opt for MPC as the advanced controller stems from its proven efficacy in addressing complex control challenges, offering enhanced adaptability, and providing a robust framework for optimizing system performance in dynamic environments. This choice is driven by the need to assess the potential advantages and limitations of MPC in contrast to the well-established PID controller, thereby contributing valuable insights to the field of control systems engineering.

A. Crazyflie Control Architecture

To comprehend the PID control architecture, it is essential to first present an overview of the inherent control structure of the Crazyflie.

The default configuration in the Crazyflie firmware employs the Proportional Integral Derivative (PID) control system for managing all desired state parameters. In this setup, the High Level Commander (HLC) or position module transmits desired position set-points to the PID position controller. Consequently, these set-points yield desired pitch and roll angles, which are then directly forwarded to the attitude PID controller. The latter determines the desired angle rates, subsequently communicated to the angle rate controller, constituting a cascaded PID controller structure. The output of this process comprises the desired thrusts for roll, pitch, yaw, and height, which are effectively managed by the power distribution system controlling the motors. There are different levels of controls in crazyflie cascaded over four levels (1) Attitude rate (2) Attitude absolute (3) Velocity (4) Position

B. PID Control Architecture

A PID (Proportional-Integral-Derivative) [8] controller is a feedback loop mechanism used in engineering for system regulation. It adjusts inputs based on the difference between desired and actual states. Comprising proportional, integral, and derivative terms, PID controllers provide stability, reduce oscillations, and effectively respond to dynamic changes in various applications, ensuring precise control. The control gain for a PID controller can be adjusted with three tunable parameters. *Kp* (proportional gain) influences the system's response to the current error, with higher values intensifying corrective actions, *Ki* (integral gain) addresses accumulated errors over time, crucial for eliminating steady-state discrepancies, and *Kd* (derivative gain) anticipates future errors by considering the rate of error change, aiding in dampening oscillations and enhancing system responsiveness.

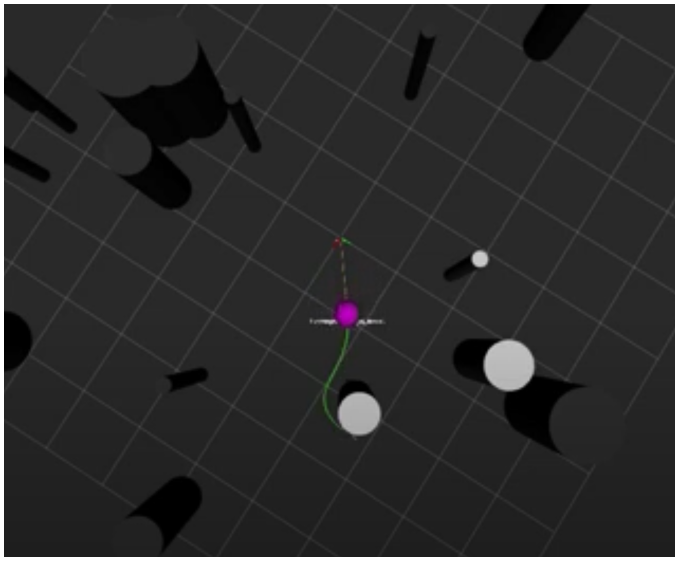

Fig. 2. Simulation framework for MPC development

The custom PID control architecture builds upon the capabilities of the built in controller. The reason to develop a custom PID control loop was to develop a baseline controller and have more control over the tuning parameters. The hardware implementation of this architecture is defined in subsequent sections.

C. MPC Control Architecture

MPC (Model Predictive Control) is an advanced control method used in dynamic systems. It predicts future system behavior through a dynamic model, optimizing control inputs iteratively. This approach, considering constraints and performance criteria, enables MPC to achieve precise control in real-time, making it valuable for applications like robotics and process control. For the scope of this project, we decided an NMPC (Non-Linear MPC) control strategy which is an extension of MPC with the ability to handle non-linear dynamics which is the case in this project. As described in Section III, we use the full non-linear dynamics to enable maximum aggression. In the subsequent sections, we talk about how the controller is implemented on the base station.

V. Simulation

For the simulation environment, we use a gazebo simulation based on the RotorS simulator [3]. The obstacles are encoded as rigid body cylinders in the environment. Fig 2 shows the MPC running in simulation on the RotorS framework. Please note that physical properties like the mass and moment of inertia of the quadrotor are configured to be the same as the RotorS quadrotor URDF.

VI. Implementation and Testing

The implementation section of this report essentially deals with the discussion of four major sections. We discuss the soft-

ware architecture, hardware architecture, payload fabrication, trajectory generation and finally controller implementation. It also intends to detail the changes made to the existing crazyflie firmware and the iterative changes made to the payload design in order to develop a viable payload. Here we also intend to discuss the tests we conducted in order to validate our design and implementations.

A. Software Architecture

The primary software architecture for this system is ROS. We wanted to ensure that there is a significantly less delay while ensuring steady communication with the drone over the crazyradio. ROS (Robot Operating System) [9] is an open source software development kit for robotics applications.

ROS offers a standard software platform to developers across industries that will carry them from research and prototyping all the way through to deployment and production. Hence choosing ROS was an important mode of software communication architecture in this project.

The software architecture consists of four main ROS nodes which essentially communicate with each other in order to establish a routine which can be thoroughly repeated to conduct easier tests and ensure least communication delays.

The four main nodes of communication are as follows: (A) Master Node, (B) Controller Node, (C) Visualization Node, (D) Planning Node. They have been explored in detail below.

1) Master Node: This node takes care of the entire mission propagation and execution. Essentially the function of the primary node is to take as input the current drone's pose by subscribing to the mocap-external-position topic and feed this pose to all the other nodes. Along with that providing valuable information such as mission progress is also an important feature of this node.
2) Controller Node: The primary task of this node is to take as an input the poses obtained from the primary node and also the controller interface we are trying to run. This node takes care of providing the actual control outputs to the crazyflie firmware based on the control inputs and poses (states) provided to it. A detail implementation of the control architecture is provided below in the controller implementation subsection.
3) Visualization Node: This node takes in the poses from primary node and processes appropriate visualization features on these poses. Features like ROS markers (cylinders and lines) are used in corelation with appropriate publishing ROS-message types such as navmsgs/Path() in order generate an Rviz visualization wherein we can view and visually understand trajectory deviation as well as reference trajectory (trajectory.txt) from planning node along with obstacles from mocap in the virtual environment.
4) Planning Node: This node obtains environmental data from motion capture and generates an environment map. Subsequently, the planner utilizes this information to craft a path to goal. However, since the generated plan may not be scaled to smaller timestamps, a more detailed trajectory is generated to better align with smaller time intervals.

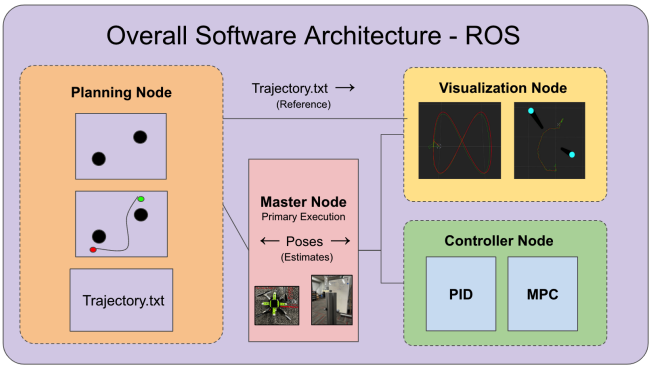

Fig. 3. Software Architecture Block Diagram

The software architecture block diagram is depicted in Figure 3

B. Hardware Architecture

The hardware architecture section deals with employing multiple hardware components such as sensors and payload in order to understand and deploy the mission requirements. The primary sensors we used to deploy multiple forms of the control architecture such as PID and NMPC along with the Mocap Marker fabrication details are as follows: (1) Flow Deck and Multiranger (2) Optitrack Mocap System (3) 3D Printed Mocap Marker Panels

1) Flow Deck and Multiranger: The initial phases of the project focused on experimenting with multiple sensor configurations and developing our controllers based on these sensor readings. The most convenient sensor which is also standardized by bitcraze for relative positioning is the FlowDeck sensor. The Flowdeck sensor outputs crazyflie's height and its position i.e. (X, Y) coordinates in relative positioning system. This implies that the zero position of the crazyflie is where it is started. Bitcraze also provides a deck/sensor attachment which can be affixed on top of the crazyflie in order to receive laser-range data. Essentially there are four laser beam sensors which deflect off of obstacles and hence, the crazyflie can detect obstacles up to an accuracy of 5 meters. Both sensors are shown in Figure 4
The plan initially entailed focusing on the usage of these sensors to develop both the controllers. It was soon observed that Flowdeck positioning data was unreliable for surfaces which do not have good features. Also, a regular drift was observed in crazyflie's positioning which often led to inaccurate state estimates and made the crazyflie converge at a location which was +-10 cms in both the coordinate directions (X and Y).
Hence, we decided to take an important decision about switching to Mocap positioning system and use the same for both, the PID and NMPC deployment. Since

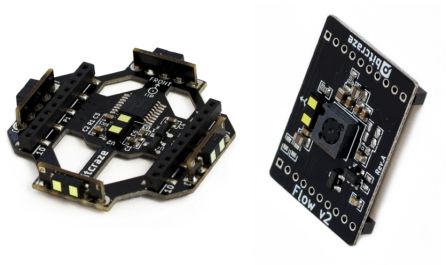

Fig. 4. Multiranger Sensor (Left) Flowdeck Sensor (Right)

Mocap provides accurate global positioning estimates, it would be highly useful as an accurate measure of input state estimate. It was also observed that Mocap's state estimates were extremely reliable and hence useful for deploying the NMPC controller.

2) Optitrack Mocap System: Motion capture (mocap) [10] is the process of recording the movement of objects or people. It involves measuring the position as well as orientation of the objects or people in physical space. The technology was originally developed for gait analysis in the life science market but is now used in a wide variety of other fields. The Optitrack Mocap system is widely used because of ultra precise pose generation with low latency and 6 DOF (Degree of Freedom) tracking for drones, ground and industrial robotics. The process simply captures the position of the object by creating a rigid body using three or more infrared reflective markers and hence tracks this rigid body across three dimensional space.
3) 3D printed Mocap Marker Panels: The 3D printed Mocap marker panel was designed and constructed while keeping certain design considerations in check. It was important to distribute the weight of this Mocap marker panel along the body of the crazyflie. Hence we decided to create a structure wherein we can distribute the weight along the entire chassis of crazyflie and hence distribute the effective load along all the motors. The Mocap marker panel (green color) is shown in Figure 5

C. Payload fabrication

Initially the design of the payload focused upon a critical consideration of utilizing flowdeck sensor. Since the flowdeck sensor is equipped with a camera that uses optiflow state estimation capturing frames of ground moved with respect to the drone, it is extremely important to ensure that there is no obstruction to the view of the crazyflie. With this in mind, an initial version of the payload was built which weighed two grams in terms of gross total weight. The first payload is depicted in Figure 6.

After switching to Mocap, we were not constrained with the payload design consideration. Since Mocap does not need to account for spatial features of the ground, the payload could be designed with no absolute constraints. The image shown in Figure 7. depicts the final iteration of the payload which overcomes certain important flaws of the previous payload. There is also a functionality to change the payload weight dynamically with the addition and removal of screws/nuts in the holes. Now the payload weight can be varied from a minimum gross weight of 3 grams and a maximum gross weight of 6 grams.

The entire payload and drone assembly is shown in Figure 8.

Fig. 5. Mocap Marker Panel (Green, PLA-20-percent Infill)

Fig. 6. 3D printed payload (1) - PLA - 20-percent Infill

Fig. 7. 3D printed payload (2) - PLA - 20-percent Infill

Fig. 8. Full assembly depicted in the image

D. Trajectory Generation

This can be categorized into two subsections as shown below:

1) Fixed Trajectories (No Obstacle Avoidance):

a) Helical Trajectory: A helical path was chosen to simulate a continuous and smooth movement in three-dimensional space. The helical trajectory was generated using Python, taking into account key parameters such as radius (r=0.5m), initial height (h=0.6m), height increments per time step (Δh=0.002m), total duration (Tf=20s), and a sampling time (Ts=0.015s). The resulting trajectory showcases a graceful ascent from the origin to a specified height, followed by a continuous helical pattern.

b) Figure 8: In addition to the helical trajectory, a figure "8" pattern was designed to introduce complexity to the robotic system's motion. The trajectory was implemented by using the parametric equations $x = r.cos(t)$, $y = r.sin(2t)$ and $z = h_0$ (h_0=0.6m) This choice of parametric equations results in sharp corners resembling the figure "8".

2) With Obstacle avoidance: Environmental data is acquired from a motion capture system and a YAML file is generated that functions as input for the Rapidly Exploring Random Trees (RRT) planner [11]. Subsequently, the planner utilizes the data provided in the YAML file to formulate a path plan using the RRT planning algorithm. Recognizing that the initial plan may not be adequately scaled for smaller timestamps, an additional step is incorporated. Specifically, the planning node undertakes the task of refining the initially broad path plan into a more detailed trajectory, ensuring better alignment with smaller time intervals.

E. Controller Implementation

As mentioned above, one of the goal of this project is to generate a comparative analysis between PID and MPC controller.

1) PID Control: To establish a baseline PID controller, our development process began with the creation of a basic code, leveraging functions from the built-in MotionCommander() class. The goal of this activity was to guide the crazyflie from point A to B. We utilized the Flowdeck for positioning. The error for this PID controller was calculated as the simple difference between the desired position and the current position. This error was then fed into a custom PID function designed to minimize deviations. The PID controller's output consisted of velocity set points, subsequently conveyed to the Crazyflie using its native commands. We implemented three distinct PID controllers for each of the three axes (X, Y, Z), as we observed that a single PID control parameter did not yield satisfactory results across all three directions. The control architecture of PID can be referred from fig 9

Furthermore, we incorporated obstacle avoidance logic using PID in our system. Obstacle detection was managed by the multiranger sensor. To execute this task, we implemented a controller switching strategy. Initially, the Crazyflie utilized the basic control loop, as described earlier, to navigate from point A to B. Simultaneously, the multiranger continuously monitored for the presence of obstacles within a predefined threshold. Upon detecting an obstacle, the code transitioned to an alternate PID control loop, aiming to minimize the inverse of the distance to the obstacle. This involved navigating either right or left based on the presence of obstacles in those directions until the multiranger signaled the absence of obstacles. Subsequently, the code reverted to the original control loop to resume the journey to point B.

However, as discussed previously upon executing the code, consistent deviations were observed in the Crazyflie's position, rendering it unable to reach the designated locations as intended. Further investigation and literature review revealed that the accuracy of the flowdeck is susceptible to external factors such as lighting conditions and floor texture. Given the project's requirement for precise Crazyflie positioning, crucial for payload pickup, it became evident that relying on flowdeck for position estimate would be insufficient. Multiple trials substantiated the need for an alternative approach to attain the necessary precision. To overcome this crucial issue to obtain correct estimates, we shifted to a motion capture system as mentioned in the above section.

In this scenario, the PID control architecture remains unchanged, with the only modification being that the position estimates now originate from the motion capture system instead of the Flowdeck.

2) MPC control: To implement an offboard NMPC, we utilized the system dynamics as mentioned in above section III with the ROS architecture as mentioned in software architecture section. ACADOS solver is used to generate fast C code for running the optimization problem on our base station computer. The base station has a 6-core 2.6GHz Intel i7 CPU and 8GB RAM. In 10 The OptiTrack motion capture system publishes the mocap observations at 200Hz via the local area network (LAN). The IMU readings from the Crazyflie are streamed from the Crazyradio at 100Hz. We then fuse the state estimates using a Kalman state estimator and publish the full state at 66.67Hz. This is fed to the non-linear MPC controller which publishes the control trajectory for the N=50 horizon. Finally, we convert the motor velocity control inputs from the MPC to roll, pitch, yaw-rate and thrust which can be sent to the Crazyflie motion commander using send_setpoint() command. The low-level controls are handled by Crazyflie's

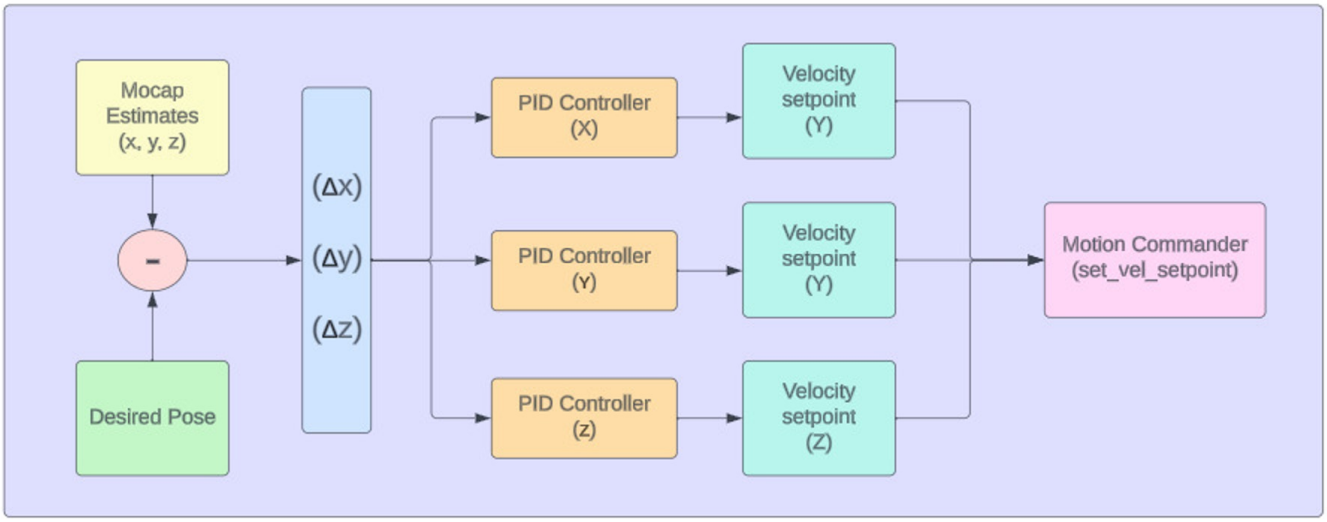

Fig. 9. Image shows a block diagram of the PID framework

built in cascaded PID controller.

F. Testing

To test the trajectory tracking, 8 scenarios have been designed with 4 scenarios tested on each of the MPC and PID controllers. These 8 scenarios are listed as follows and are self-explanatory,

- PID with Custom Trajectory and without payload
- PID with Custom Trajectory and with payload
- PID with Obstacle Avoidance and without payload
- PID with Obstacle Avoidance and with payload
- MPC with Custom Trajectory and without payload
- MPC with Custom Trajectory and with payload
- MPC with Obstacle Avoidance and without payload
- MPC with Obstacle Avoidance and with payload

VII. Demo Results

The qualitative analysis of the framework is depicted in Figure 11 wherein the NMPC framework is tracking figure 8 reference trajectory. Here the red line depicts the reference trajectory and the green is the tracked trajectory in real-time by the drone. This visualization is facilitated by the visualization node developed by us.

Similarly, the Figure 12 shows the obstacle avoidance trajectory tracking performance wherein the blue columns depict the obstacles and the red and green trajectories are reference and tracked path.

The following metrics are used to evaluate the trajectory tracking of the Crazyflie in the 8 scenarios tested. For calculation of the metrics, sufficient time sampling and backward interpolation is done to make sure the reference and tracked trajectories are of similar shape.

- Root Mean Square Error
- Maximum Absolute Deviation

1) Root Mean Square Error (RMSE):
The Root Mean Square Error calculates the square root of the average of the squared differences between the points on the reference trajectory and the tracked trajectory. The Root Mean Square Error is given by the following equations

$$\text{RMSE}_i = \sqrt{\frac{1}{N}\sum_{j=1}^{N}(i_{\text{ref}[j]} - i_{\text{traj}[j]})^2}$$
$$\text{where } i \in \{x, y, z\}$$

2) Maximum Absolute Deviation (MAD):
The Maximum Absolute Deviation calculates the Maximum Absolute Deviation observed between the points on the reference trajectory and the tracked trajectory. The Maximum Absolute Deviation is given by the following equations

$$\text{MAD}_i = \max_j |i_{\text{reference}[j]} - i_{\text{trajectory}[j]}|$$

where i $\in \{x, y, z\}$

The Crazyflie has been tested on 8 scenarios and the metrics defined above (RMSE and MAD) are summarized in Tables I and II. The Plots corresponding to certain scenarios are as shown in Figures [17 - 16].

multirow

From the table and plots, it can be inferred that MPC shows lesser error than PID with respect to Root Mean Square error and the Maximum Deviation of the Trajectory in the three directions i.e. X,Y and Z directions.

From the plots, it can be inferred that MPC takes lesser time when compared to PID to track and complete the trajectories.

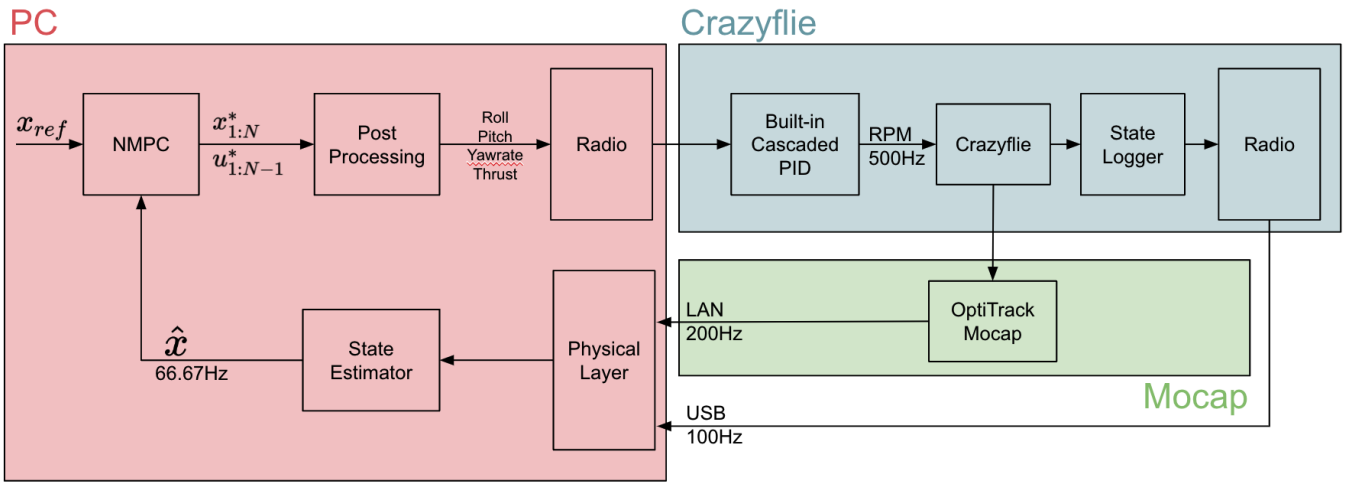

Fig. 10. Image shows a block diagram of the NMPC framework

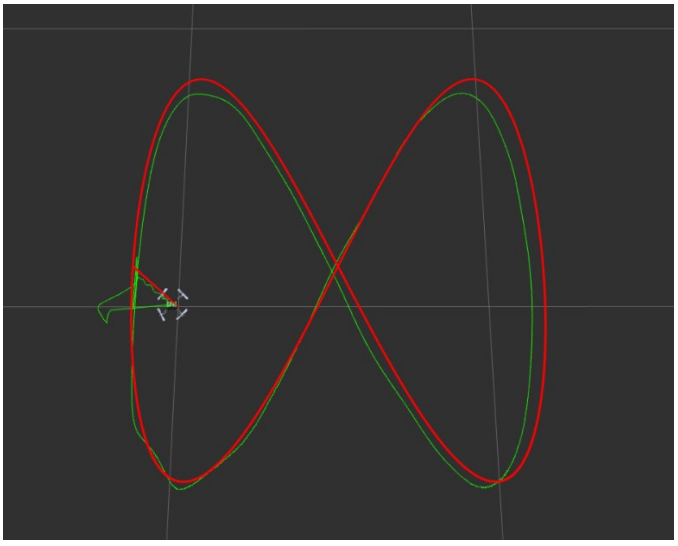

Fig. 11. MPC Custom Trajectory with Payload

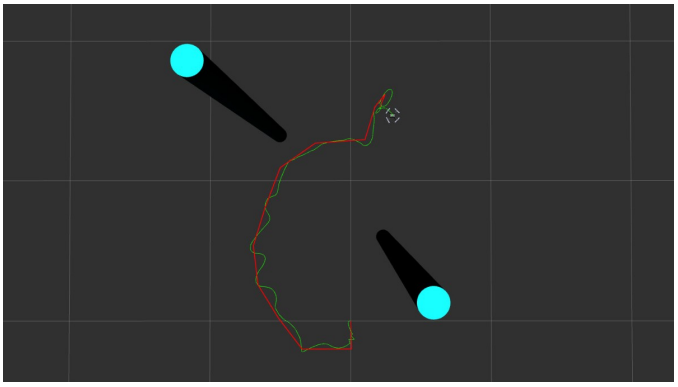

Fig. 12. MPC Obstacle Avoidance with Payload

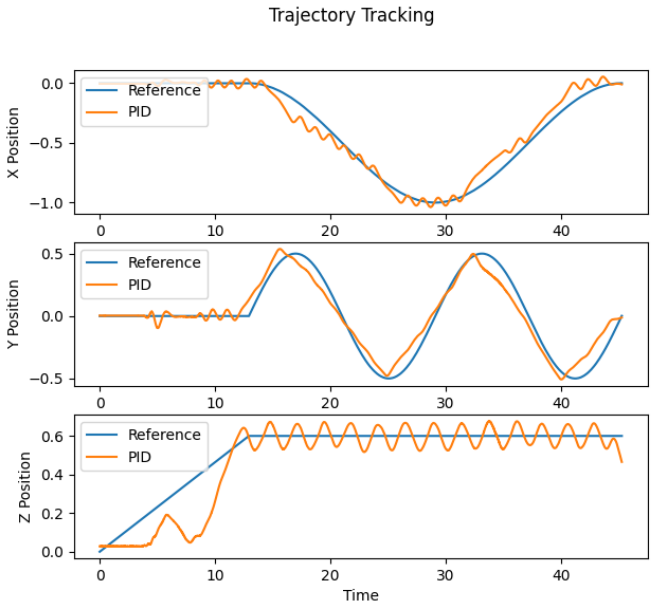

Fig. 13. PID Custom Trajectory with Payload

TABLE I
OBSTACLE AVOIDANCE

		Payload		No payload	
		PID	MPC	PID	MPC
RMSE	X	0.0468	0.0264	0.0456	0.023
	Y	0.0432	0.0274	0.0667	0.0467
	Z	0.0667	0.0401	0.1052	0.0734
MAD	X	0.1735	0.0685	0.1346	0.0312
	Y	0.1442	0.0991	0.1542	0.1069
	Z	0.3128	0.1295	0.3367	0.1357

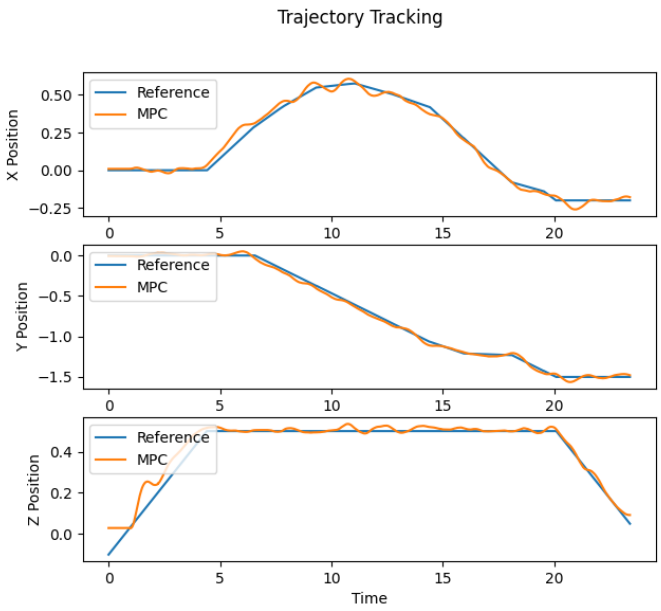

Fig. 14. MPC Obstacle Avoidance Trajectory with Payload

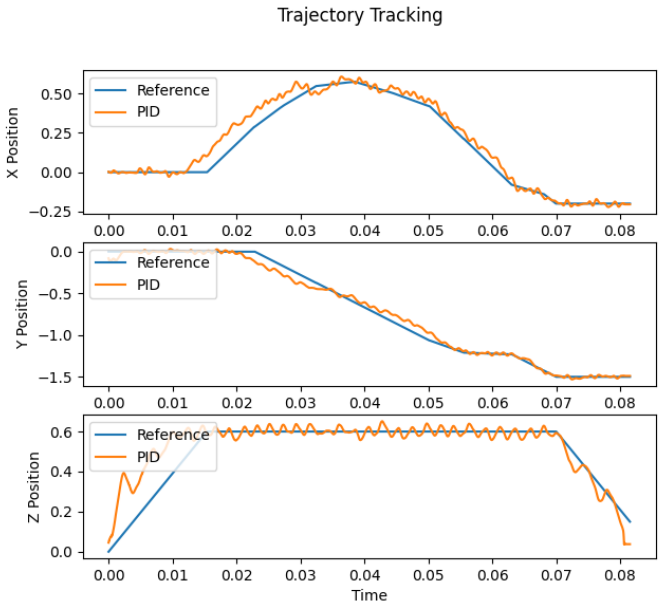

Fig. 15. PID Obstacle Avoidance Trajectory without Payload

TABLE II
CUSTOM TRAJECTORY TRACKING

		Payload		No payload	
		PID	MPC	PID	MPC
RMSE	X	0.0504	0.0456	0.0539	0.0496
	Y	0.0852	0.0597	0.1067	0.0853
	Z	0.09423	0.0569	0.1175	0.0823
MAD	X	0.1735	0.0702	0.1563	0.0624
	Y	0.1965	0.1274	0.2067	0.1534
	Z	0.3134	0.1311	0.3548	0.1474

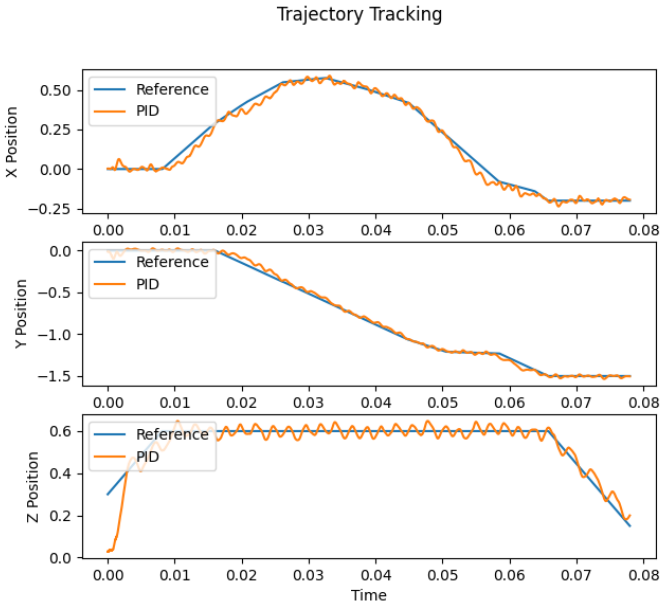

Fig. 16. PID Obstacle Avoidance Trajectory with Payload

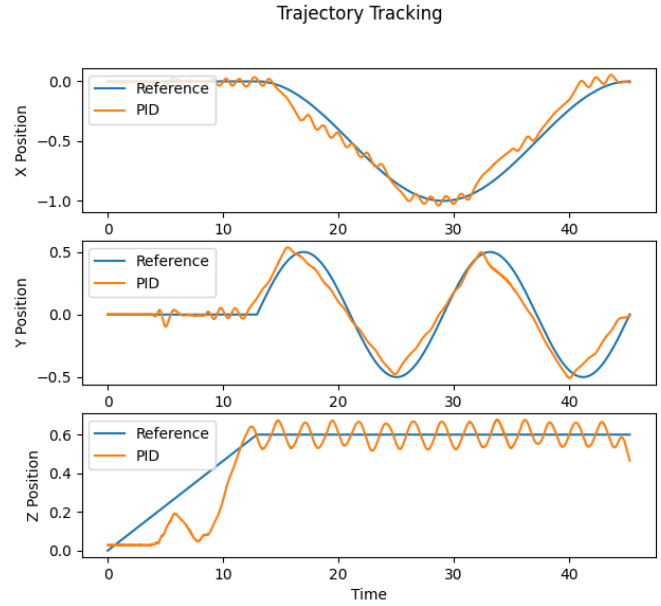

Fig. 17. PID Custom Trajectory with Payload

VIII. CONCLUSIONS

In conclusion, the Nonlinear Model Predictive Control (NMPC) algorithm has demonstrated superior performance compared to the Proportional-Integral-Derivative (PID) controller in the aspects of trajectory tracking and obstacle avoidance. The trajectory deviations observed in the MPC-controlled drone were notably fewer than those observed in the PID-controlled scenarios, leading to a more efficient and faster achievement of the goal.

Moreover, both controllers have handled payloads well, with the actual trajectory closely adhering to the reference trajectory even under the additional load. This resilience ensures that the drone maintains stability and accuracy in payload deliveries, emphasizing the practical applicability of the controllers in real-world scenarios.

In summary, the findings underscore the efficacy of MPC in trajectory tracking and obstacle avoidance, making it a favorable choice for applications that demand high precision and responsiveness.

IX. Acknowledgement

We would like to express sincere gratitude to the professor Mark Bedillion and teaching assistant Khai Nguyen whose guidance and support have been invaluable in completing this project. We would also like to acknowledge Master's students: Prakrit Tyagi and Jong Hoon Park for their support in the crucial moments of this project.

References

[1] "Jetson Nano Developer Kit — developer.nvidia.com." https://developer.nvidia.com/embedded/jetson-nano-developer-kit. [Accessed 16-12-2023].

[2] "Crazyflie 2.1 — Bitcraze — bitcraze.io." https://www.bitcraze.io/products/crazyflie-2-1/. [Accessed 16-12-2023].

[3] D. Falanga, P. Foehn, P. Lu, and D. Scaramuzza, "PAMPC: perception-aware model predictive control for quadrotors," *CoRR*, vol. abs/1804.04811, 2018.

[4] A. Alavilli, K. Nguyen, S. Schoedel, B. Plancher, and Z. Manchester, "Tinympc: Model-predictive control on resource-constrained microcontrollers," 2023.

[5] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen, and M. Diehl, "acados – a modular open-source framework for fast embedded optimal control," *Mathematical Programming Computation*, Oct 2021.

[6] H. J. Ferreau, C. Kirches, A. Potschka, H. G. Bock, and M. Diehl, "qpoases: A parametric active-set algorithm for quadratic programming," *Mathematical Programming Computation*, vol. 6, pp. 327–363, 2014.

[7] B. B. Carlos, T. Sartor, A. Zanelli, G. Frison, W. Burgard, M. Diehl, and G. Oriolo, "An efficient real-time nmpc for quadrotor position control under communication time-delay," in *2020 16th International Conference on Control, Automation, Robotics and Vision (ICARCV)*, pp. 982–989, 2020.

[8] Y. Li, K. H. Ang, and G. Chong, "Pid control system analysis and design," *IEEE Control Systems Magazine*, vol. 26, no. 1, pp. 32–41, 2006.

[9] "ROS: Home — ros.org." https://www.ros.org/. [Accessed 16-12-2023].

[10] J. S. Furtado, H. H. Liu, G. Lai, H. Lacheray, and J. Desouza-Coelho, "Comparative analysis of optitrack motion capture systems," in *Advances in Motion Sensing and Control for Robotic Applications: Selected Papers from the Symposium on Mechatronics, Robotics, and Control (SMRC'18)-CSME International Congress 2018, May 27-30, 2018 Toronto, Canada*, pp. 15–31, Springer, 2019.

[11] S. LAVALLE, "Rapidly-exploring random trees : a new tool for path planning," *Research Report 9811*, 1998.