Accurate and asynchronous trajectory primitives for micro aerial vehicle navigation

John Wallace
Department of Electrical and Computer Engineering
Princeton University
jw0535@princeton.edu

Nathaniel Simon
Department of Mechanical and Aerospace Engineering
Princeton University
nsimon@princeton.edu

***Abstract*—Micro aerial vehicles (defined as unmanned aerial vehicles weighing less than 100 g) suffer from a highly constrained payload capacity, preventing the use of traditional depth sensors and effective autonomous navigation. MonoNav is a novel navigation stack (developed by the Princeton IroM Lab) that makes use of a neural network for depth estimation, combined with standard reconstruction and path planning approaches in a modular architecture. This enables a Crazyflie 2.1 to navigate autonomously (at 37 g total weight) using depth information from a light-weight monocular camera, offboard computation, and a motion primitive planner. This planner does not make use of the Crazyflie's built-in high-level commander that employs positional feedback and a fast Mellinger controller to follow predefined trajectories asynchronously. This is due to a firmware drawback that prevents these trajectories from being initiated relative to the drone's current position. In this work, the Crazyflie firmware is modified to allow relative, in-flight initiation of these closed-loop trajectories. It is shown that the closed-loop trajectories are significantly more precise and accurate than the previous implementation of primitive following, and their asynchronous nature allows for higher-latency batch-processing of multiple images. Combined, these benefits enable autonomous navigation of challenging environments at higher significantly speeds than previously possible: from 0.5 m/s to 0.9 m/s (80% increase).**

I. Introduction

Due to their small size, micro aerial vehicles (MAVs – under 100 g) are ideal for agile exploration of a range of environments, particularly indoors. This size, however, greatly limits the available payload, affecting their sensing and computation abilities. Standard autonomous UAVs rely on sensors (e.g., depth cameras and LiDAR scanners) that can weigh more than a MAV itself. While there has been some advancement in processors and sensors suitable for MAVs [1,2], they are still fundamentally limited by the scale of the platform. MonoNav, a novel open-source navigation stack for MAVs developed by the Intelligent Robot Motion (IRoM) Lab at Princeton, enables navigation using depth estimation and reconstruction with a monocular camera and an off-the-shelf depth estimation neural network (ZoeDepth; Fig. 1)[3,4]. The neural network inference requires a large GPU and is performed offboard, while a low-cost, light-weight monocular camera is housed on a Crazyflie 2.1 drone frame (37 g total). A key benefit of the MonoNav stack is its modular nature, allowing navigation modules to be interchanged and improved independently. For instance, the depth estimation network can be paired with any planner that makes use of depth information. Each module can then be improved independently with standard estimation or planning techniques. The current iteration of MonoNav uses a motion primitive path planner, which repeatedly selects a predefined curvature to follow based on a reconstructed depth map.

Currently, MonoNav executes primitives with an open-loop control method by commanding a sequence of velocity and yaw rate setpoints. These setpoints are defined by constant forward velocity and a half-sine yaw rate profile (to ensure constant yaw at the beginning and end of each trajectory for smooth chaining of primitives). MonoNav uses a set of a few primitives with a range of evenly-spaced maximum yaw rates, corresponding to a range of curvatures which can be selected. An example library of five primitives is shown in Fig. 2. These setpoints are used by the Crazyflie's built-in cascaded PID controller to control its motors, but there is no feedback to vary the velocity setpoints based on the vehicle's positional error. A small amount of lateral velocity is commanded throughout the trajectory to better match the primitive, which was only tuned for a conservative 0.5 m/s forward velocity. These primitives are restricted to this speed and simple geometry, preventing MonoNav from being made more agile with faster and more complex primitives.

Moreover, this primitive following runs synchronously in the same program as the depth estimation, preventing longer batch inferences of multiple images from being performed (the Crazyflie requires velocity setpoints at to be sent at a rate of between 5 and 10 Hz to maintain a hover, otherwise it will land). In addition, in order to run ZoeDepth, the vehicle's camera image must be transformed to match the intrinsics of the camera used to train the network, causing much of the periphery to be lost. This can be solved by segmenting the capture into smaller images arranged horizontally and transforming each individually, but this leads to a higher computation time per inference. Batch-inferencing decreases the rate at which computation time scales with the number of images, but the processing of wide-angle data is still slow.

The Crazyflie platform does support the asynchronous execution of high-level trajectories uploaded to the drone's memory before flight. The platform is composed of a firmware C codebase, and a Python library to send commands to the firmware. Trajectories are uploaded and initiated using a Python client, and executed within the drone's firmware. These trajectories are represented as three seven-degree polynomial

Fig. 1. The MonoNav platform (left). A vizualization of a MonoNav camera image and a resulting depth estimation.

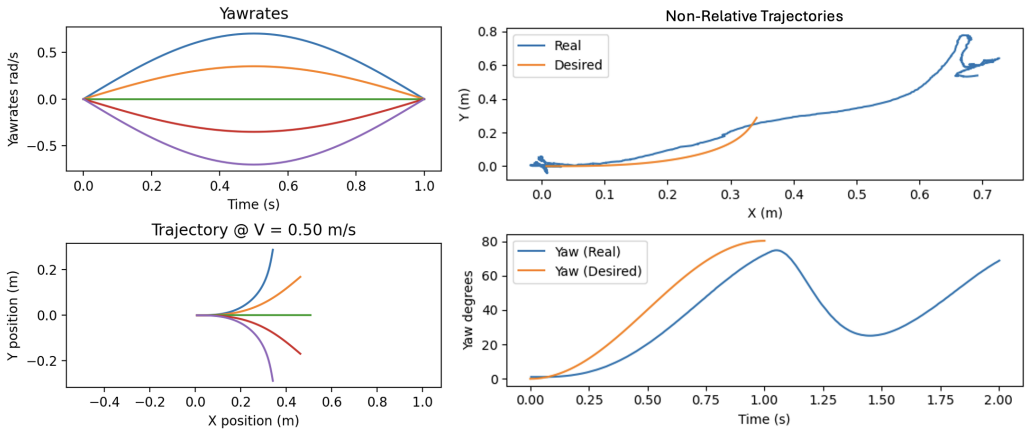

Fig. 2. A library of five motion primitives of varying half-sine yaw rate profiles (left). Two chained trajectories demonstrating the non-relative yaw (right; courtesy of Nathaniel Simon). The second trajectory starts at the final position of the first trajectory, but its yaw is not initiated relatively.

functions of time for each of the drone's two positional dimensions and yaw. The trajectories are generally intended for longer predefined behaviors, where the drone's entire flight is defined by a small number of trajectories, rather than the more dynamic behavior of MonoNav. This made it challenging for the high-level trajectories to be used in MonoNav, due to the requirement of initializing a new trajectory relative to the vehicle's current position and yaw. While there is an option to run the trajectories relatively, this was only with respect to position (and not yaw). When initiated, these trajectories would begin at the endpoint of the previously-ran trajectory but would command it to reorient toward the drone's global yaw origin, as shown in Fig. 2.

These trajectories use the Crazyflie's "High-level commander" to command positional setpoints (rather than velocity, as used in the original MonoNav version). The drone's firmware supports a Mellinger controller to follow these high-level setpoints, which is faster than the standard cascaded PID controller [5]. The Mellinger controller determines the required motor speeds to achieve a positional setpoint using the Crazyflie's known dynamics, and was designed to follow trajectories like the motion primitives used by MonoNav. These benefits can be achieved by modifying the drone's firmware to enable fully-relative, asynchronous trajectories. This will improve MonoNav's performance by enabling more agile primitives, improving the precision and accuracy of their execution, and enabling higher latency batch-inferences. Previous MonoNav work shows that this novel stack is possible, but much improvement is needed before MonoNav can operate with agility in complex environments.

II. Achieving Asynchronous, Closed-Loop Primitives

As stated, trajectories are represented in Crazyflie firmware as polynomial functions of time. When a trajectory is initiated, these functions are evaluated for each timestep to generate the position and yaw trajectories. For relative position, the original firmware would add a constant term to these polynomials equivalent to the ending position of a previous trajectory at the time of a new trajectory's initiation. This is not sufficient to chain trajectories, as each new trajectory's yaw would begin at the global yaw origin. To achieve relative yaw, the positional output of the polynomials are rotated about the vehicle's initial position by its starting yaw, which was also added as a constant term to the yaw polynomial. After this rotation, the outputs are shifted by the initial position. With these changes, trajectories can be chained effectively, with each trajectory defining the vehicle's movement relative to its starting point.

Another firmware modification was made to improve the accuracy of the trajectory chaining. Because new trajectories are initiated at a fixed rate, set by the duration of each trajectory before runtime, it is common for the vehicle to

not fully complete its current primitive before a new one is initiated. To prevent a buildup of error from a large number of incomplete trajectories, the firmware was modified to initiate a new trajectory at the drone's current position rather than the end of the previous trajectory. These changes enable a much more dynamic use of the trajectories, where trajectories can be chained with in-flight replanning and trajectory initiation. This is used to replace MonoNav's previous motion primitive implementation, and will be proposed as a merge to the Crazyflie's official firmware version.

III. Comparison to Open-Loop Primitives

To evaluate the benefits to MonoNav provided by the high-level primitive execution, a number of tests were performed to compare them with the previous open-loop control. Moreover, the high-level control was used to run MonoNav at a higher speed than initially achieved. In single-primitive tests, the primitives were assigned a duration of $0.5/v$ s (where v is the primitive's forward velocity) to maintain a consistent 0.5 m distance traveled per primitive. When the entire MonoNav stack was run, the primitives were given a duration slightly longer than the estimation and planning computation times (0.6 s). This also allows the primitive library to be defined based on the tightest possible turn a drone can execute at a given velocity (the curvature of the primitives would remain constant with increasing velocity if their duration is not decreased). To allow the drone to reach the desired forward velocity, each test was initiated after a straight-line "start-up" trajectory with a forward velocity and duration matching that of the test's primitives.

First, the precision and accuracy of the primitive execution is tested. To do so, a library of five primitives (Fig. 2) with 0.5 m/s forward speed was evaluated three times: once under the original open-loop control, once using closed-loop PID control, and once using closed-loop Mellinger control. The primitives were also run at 0.75 m/s and 1.0 m/s with closed-loop Mellinger control to test the high-speed performance. For each test group, the evaluations were run five times (with the exception of the closed-loop PID control, which was run three times). These trials are shown in Fig. 3. The mean squared error and its standard deviation were calculated for each test group (combining every primitive and trial, and averaging over the entire test group), and are reported in Table 1. The error was considered to be the distance between a setpoint and the drone's actual position at a particular time in the trajectory. This metric combines notions of positional and time deviation from the desired primitive, since the error is calculated with respect to time. As a result, two trajectories might have the same mean squared error if one completes the whole primitive but is positionally inaccurate, while another is highly positionally accurate but does not finish the primitive.

Interestingly, the PID control of the closed-loop primitives is worse than the open-loop primitives. This is due to the slow speed of the PID controller, which is unable to reach the desired yaw rate before the trajectory finishes and the drone is commanded to land. It is also unable to reach the desired forward velocity before the start-up trajectory finishes, so it visibly pauses when the primitive is commanded. This pause can be fixed by allowing the new primitive to begin at the start-up trajectory's final setpoint, but the drone is still unable to reach peak yaw rate. Importantly, the Mellinger controller is significantly more accurate and precise than the original open-loop primitives. This controller's performance is much tighter around the desired primitives, having the lowest error and standard deviation of all the control modes at 0.5 m/s. The error and standard deviation is highest at 1.0 m/s (0.75 m/s has comparable accuracy to 0.5 m/s), indicating an expected tradeoff between velocity and accuracy. Still, the accuracy at this speed is not prohibitively low, and 1.0 m/s forward velocity would be entirely unattainable with the previous implementation.

Finally, the entire MonoNav stack was run using the closed-loop primitives (with batch-inferencing of the wide-angle captures) and the Mellinger controller at 0.5, 0.75, and 0.9 m/s forward velocity. These tests were run in a hallway approximately 1.5 m wide with a 90 deg left turn, with the reconstructed environments shown in Fig. 4. These runs demonstrate the chaining of asynchronous primitives at higher speeds than previously possible. This, combined with the Mellinger controller's effectiveness, enabled MonoNav to quickly navigate an environment that was previously challenging at slower speeds. It was found that MonoNav would come too close to the walls when run at 1.0 m/s forward velocity. With shorter depth-estimation times, a higher replanning frequency, and shorter primitive durations, the drone would likely have been able to replan fast enough to avoid the walls before aborting. This indicates that MonoNav is being limited by the speed of the depth-estimation (in its current implementation) in this environment.

IV. Conclusion

The relative yaw firmware modification brings clear benefits to MonoNav's navigation which unlock higher performance in a number of ways. First and foremost, the use of the Crazyflie's built-in asynchronous trajectory framework allows the use of infinitely-customizable primitives. This means primitives of any forward velocity, duration, or form can be defined according to a user's needs. Because path execution is now handled by a specialized controller, it is reliably precise at a variety of primitive definitions. Moreover, the ability to initiate primitives asynchronously is immensely useful from a practical standpoint. It cleans up the main MonoNav code and allows longer processes—such as batch inferencing—to run without interrupting to maintain a trajectory.

These changes further demonstrate the success of MonoNav's modular approach to MAV navigation. Motion primitive path planning is a standard navigation method, and any vehicle with a microcontroller can support trajectories defined in this manner. These are standard navigation methods independent of MonoNav's depth estimation. Such an architecture allows the navigation and control modules to be separated

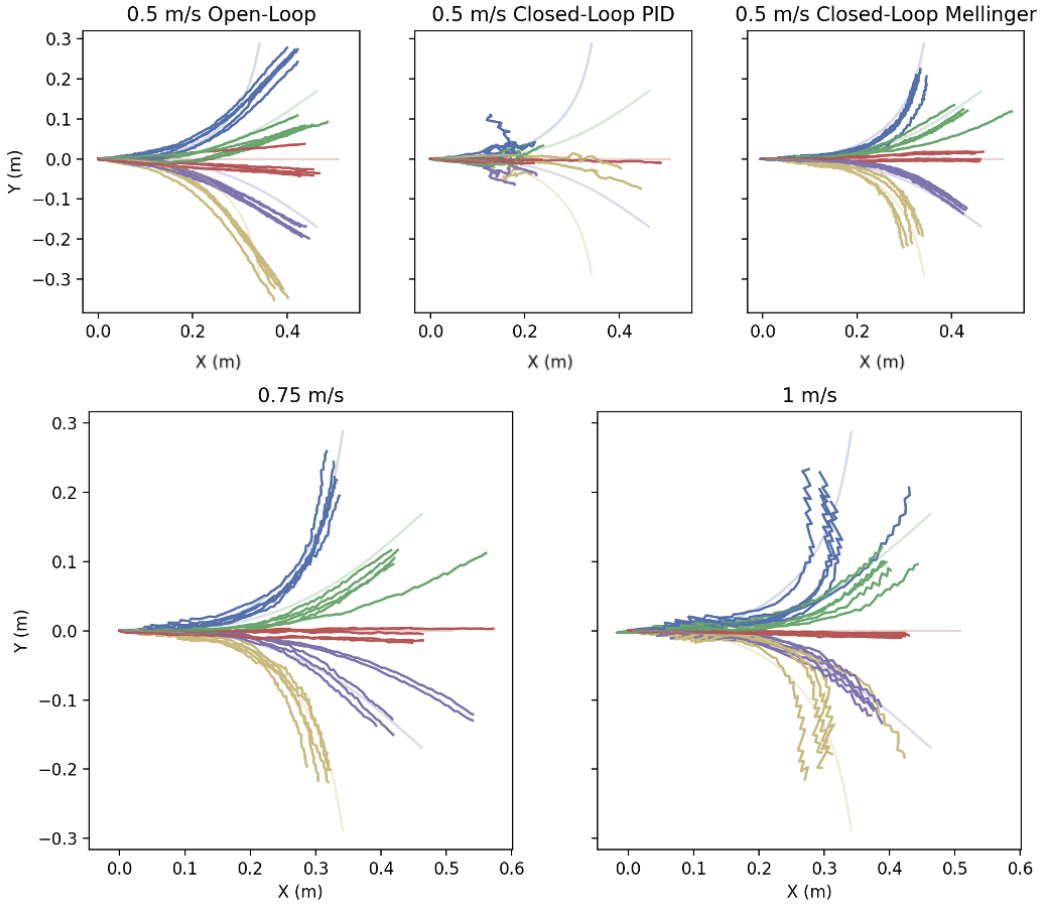

Fig. 3. Primitive-following performance of various speeds and control methods.

	0.5 m/s Original	0.5 m/s PID	0.5 m/s Mellinger	0.75 m/s Mellinger	1.0 m/s Mellinger
MSE (m)	0.014	0.019	0.011	0.010	0.014
SD (m)	0.021	0.026	0.012	0.010	0.012

TABLE I
MEAN SQUARED ERROR AND ITS STANDARD DEVIATION FOR VARIOUS SPEEDS AND CONTROL METHODS.

and improved using methods that may not be possible if the two were coupled, as in end-to-end approaches.

There are many improvements that can be made across the stack to better MonoNav's performance. Primarily, there is likely a better scheme for using memory of an environment to inform MonoNav's path planning. In the original MonoNav, all depth estimations were integrated into a 3D reconstruction and used to inform future planning. Due to inaccurate past estimations causing valid primitives to appear invalid, this could be removed because wide-angle estimation is sufficient to navigate without memory. This approach has inefficiencies, as it ignores any previous useful information. An ideal solution would implement some type of memory that does not allow noisy estimations to interfere with navigation.

Additionally, MonoNav is limited by the inference time of depth estimation. A simple improvement would be the quantization of the neural network, to reduce the precision of its output and activation values (from float to int, for example) and speed up estimation by up to four times. It is generally not necessary to know the precise location of an obstacle to navigate (in theory an "adaptive" quantization technique could be used to obtain slower, higher precision estimations when necessary). This would allow for faster estimation and planning, but also move toward lighter estimation networks and onboard computation. Finally, there are a host of hardware improvements that would improve MonoNav's performance without modifying the stack. Higher quality digital cameras would reduce noise and artifacts passed to the estimation network. Cloud connectivity will free MonoNav from the current short-range radio link required for offboard computation, and allow it to navigate wherever there is reception.

V. ACKNOWLEDGEMENTS

I am thankful first and foremost to Nathaniel Simon, for his guidance throughout this project and his ingenuity in creating

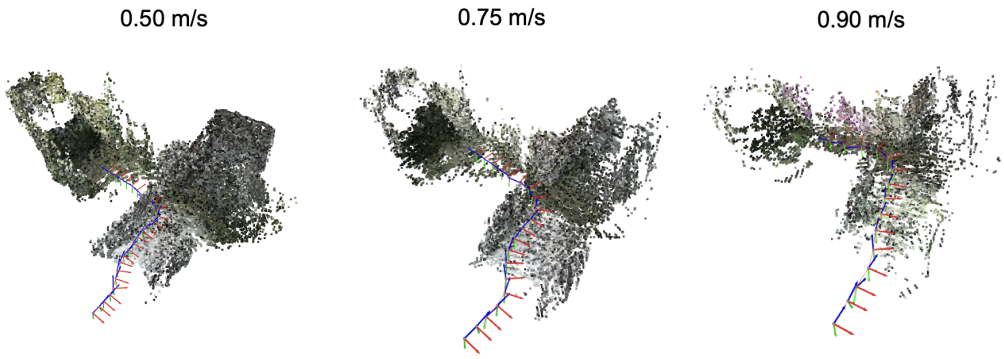

Fig. 4. Reconstructions of MonoNav navigating through a challenging turn at higher speeds using the closed-loop trajectories. Note that the combined reconstruction is not what MonoNav uses for navigation (only the current depth estimation is used). Also note that the density of depth estimates throughout the entire navigation decrease as the speed increases.

MonoNav. I am thankful to Bitcraze for their wonderful open source product and the support they provided to MonoNav. And of course, I am thankful to Professor Majumdar and the rest of IRoM Lab for taking me into the group. Also, thanks to the Princeton University ECE department, and especially Lori Bailey, for their assistance in the independent work program.

REFERENCES

[1] D. Palossi, F. Conti, and L. Benini, “An Open Source and Open Hardware Deep Learning-powered Visual Navigation Engine for Nano-UAVs,” 2019 15th International Conference on Distributed Computing in Sensor Systems (DCOSS), Santorini, Greece, 2019, pp. 604-611, doi: 10.1109/DCOSS.2019.00111.

[2] V. Niculescu, H. Müller, I. Ostovar, T. Polonelli, M. Magno and L. Benini, ”Towards a Multi-Pixel Time-of-Flight Indoor Navigation System for Nano-Drone Applications,” 2022 IEEE International Instrumentation and Measurement Technology Conference (I2MTC), Ottawa, ON, Canada, 2022, pp. 1-6, doi: 10.1109/I2MTC48687.2022.9806701.

[3] N. Simon and A. Majumdar, “MonoNav: MAV Navigation via Monocular Depth Estimation and Reconstruction,” 2023, doi: arXiv:2311.14100v1.

[4] S. F. Bhat, R. Birkl, D. Wofk, P. Wonka, and M. Müller, “ZoeDepth: Zero-shot Transfer by Combining Relative and Metric Depth,” 2023, doi: arXiv:2302.12288.

[5] D. Mellinger and V. Kumar, ”Minimum snap trajectory generation and control for quadrotors,” 2011 IEEE International Conference on Robotics and Automation, Shanghai, China, 2011, pp. 2520-2525, doi: 10.1109/ICRA.2011.5980409.